HAL
open science

Contributions to pattern set mining : from complex datasets to significant and useful pattern sets

Tatiana Makhalova

To cite this version:

Tatiana Makhalova. Contributions to pattern set mining : from complex datasets to significant and useful pattern sets. Computer Science [cs]. Université de Lorraine, 2021. English. NNT : 2021LORR0124 . tel-03342124

HAL Id: tel-03342124

https://hal.univ-lorraine.fr/tel-03342124

Submitted on 13 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE LORRAINE

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

École doctorale IAEM Lorraine

Contributions à la fouille d'ensembles de motifs : des données complexes à des ensembles de motifs signifiants et réutilisables

THÈSE

présentée et soutenue publiquement le 23 juin 2021

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Tatiana Makhalova

Composition du jury

Président :	François Charoy	Pr. Université de Lorraine, Nancy, FR
Rapporteurs :	Arnaud Soulet	MCf HDR, Université de Tours, Tours, FR
	Jilles Vreeken	Pr. Centre de cyber-sécurité CISPA, Saarbrüucken, DE
Examinateurs :	Antoine Cornuéjols	Pr. AgroParisTech, Paris, FR
	Elisa Fromont	Pr. Université de Rennes, Rennes, FR
	Esther Galbrun	CR Inria, Université de l'Est de la Finlande (UEF), Kuopio, FI
	Christel Vrain	Pr. Université d'Orléans, Orléans, FR
Directeurs :	Sergei O. Kuznetsov	Pr. Université nationale de recherche EHESE, Moscou, RU
	Amedeo Napoli	DR CNRS, LORIA, Nancy, FR

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

With this dissertation, I have finished a big and very important part of my life. I have given all my enthusiasm, put a lot of effort to arrive at this point, but it would not have been possible without a huge number of people who were with me throughout everything. I am definitely lucky to meet them. The people who give me love, support, comfort, passion for what I do, learn me a sense of integrity and decency. I am very thankful to all of them for guiding me — consciously or not — in difficult moments, for offering me the strength to continue when I am falling, for sparking interest in science, for making my life bright and pleasant.

I would like to thank my school mathematics teacher Elena S. Parkacheva, who showed me the beauty of mathematics, my first scientific supervisor Sergei V. Rusakov, who showed me that science is an excellent tool to make this world better. I thank him and Olga L. Rusakova for showing me how human science and everything around it can be. Thanks to the professors of the Perm State University, the Higher School of Economics, and the Blaise Pascal University, where I did my studies, for giving me a solid basis for my further research. I also thank Kirill Yurkov, a very enthusiastic researcher who has excited my interest in computer science. I will never forget the summer schools where he taught us so many things.

I want to thank my close friends Ksenia Plekhova, Alena Korotaeva, Natalia Korepanova for sharing nice moments at school, university, work and just in life, for the memories that make me smile till now. I would like also to thank the best roommates Zhanar Bekmurzina and Rimma Karapetyan for making our university flat the warmest place in Moscow, the place where I wanted to return every evening.

When I arrived in Nancy for doing my doctorate I met so many amazing people with whom I started this journey. I send the warmest thanks to my friends Anastasia Shimorina, Anna Liednikova, Veronika Bolshakova for sharing everyday life, for supporting me, and for organizing our lovely Friday evenings.

Very special thanks to my very close friend Anastasia Tsukanova. It is very dear to me that I can share with you everything! Thank you for our long walks in the picturesque surroundings of Nancy, for interesting conversations, for letting me see the world differently, for helping me to overcome a lot of things that I could not be able to overcome alone. Thank you for being here in the funniest and saddest moments.

I thank my kind colleagues from Loria with whom I spent a lot of time preparing this thesis. I keep very pleasant memories about our Orpailleur team: Alexandre Bazin, Quentin Brabant, Adrien Coulet, Sébastien Da Silva, Nyoman Juniarta, Esteban Marquer, Pierre Monnin, Lauréline Nevin, Justine Reynaud, Claire Theobald, Yannick Toussaint, Georgios Zervakis, Laura Alejandra Zanella-Calzada, as well as the colleagues from the other teams: Antoine Chemardin, Kévin Dalleau, Iordan Iordanov, Anna Kravchenko, Thomas Lacour, Dominique Mias-Lucquin, Hans-Jörg Schurr, Athénaïs Vaginay. Thank you for your enormous support, for our fun lunches, fascinating conversations, awesome dinner parties. Thanks to my amazing "co-bureau" Nacira Abbas, Guilherme Alves, and Huber Laurine for creating a wonderful and magnetic ambiance, for making working hours bright, for becoming very good friends to me! A special thanks to our team leaders Miguel Couceiro and Amedeo Napoli for radiating endless positive, warmth, and friendliness. Thank you for organizing online evenings during the COVID time. These evenings together with your kind attention to everyone replaced the sense of isolation and loneliness with pleasant memories, jokes, and playing musical instruments.

I thank Larisa I. Antropova, Souad Boutaguermouchet, Veronique Constant, Emmanuelle Deschamps, Sabrina Ferry-Tritz, Blanca Ferdally Jacobs, Sergey S. Silantiev for their kind assistance with all documents and bureaucracy, thanks to Jonathan Alcuta and Patrice Ringot for technical support.

Special thanks to my co-authors Aleksey Buzmakov, Dmitry Ilvovsky, Boris Galitsky, Martin Trnecka, my master's degree supervisor Lhouari Nourine, and the researchers from the FCA community for our fruitful discussions, interesting research projects, nice conferences and workshops that we enjoyed a lot together.

I would like to thank the members of the thesis committee François Charoy, Antoine Cornuéjols, Elisa Fromont, Esther Galbrun, Arnaud Soulet, Christel Vrain, Jilles Vreeken, who kindly agreed to revise my thesis. Thank you for your thorough works, and valuable comments on my research and an interesting discussion during the thesis defense.

This thesis would not have been possible without my academic supervisors Sergei 0. Kuznetsov and

Amedeo Napoli. Thank you a lot for your tremendous efforts, for your high standards that prompted me to do my best, for your passion for science and for your professionalism, which inspired me a lot, for our enjoyable meetings full of science and fun. Thank you for taking care of me so much, for being so much more than academic supervisors. Thank you a lot for opening many doors to me and for introducing me to very interesting people, for guiding me in my research and in my life. Sergei, thank you for making Moscow a friendly city to me, thank you for making everything to provide a comfortable environment for research. Amedeo and Isabelle, thank you for becoming my second family here in France. It was not easy to live so far away from my family, and you gave me so much care and love, your kind attention, and your precious support. Thank you for the unforgettable dinners and Christmas evenings, which made me feel at home, and for the amazing fanfare performances, which I loved a lot.

I want to say the warmest thanks to my family that I missed a lot here. Thanks to my closest relatives that worried about me, encouraged me, support me, and just giving me the feeling that they were always next to me. Thanks to my uncle Oleg, aunts Eugenia and Maria, your messages and calls warmed my heart and made me smile a lot. Thanks to my cousins Anna, Vitaliy, Tatiana for encouraging me, for making me feel better every time I saw your faces on the screen. With great love, I thank my dear parents and sister. Mommy, daddy, my sweet sister Lena, thank you for being with me all this time! Thank you for your immense love, infinite care, for your enormous support, for not letting me lose heart when I was ready to do it. Thank you for being with me every second of my life even staying thousands of kilometers away. Thank you for doing everything you could to make me happy. Mommy, daddy, Lena, and David, thank you for waiting for me in your warm houses, for giving me days full of love, fun, and lovely family moments. Finally, I thank those people that I, unfortunately, forgot to mention.

Contents

Chapter 1
Introduction

1.1 Outline of pattern mining . . . 1
1.2 Thesis structure and contribution . . . 3

Chapter 2
Background

2.1 Formal concept analysis . . . 7
2.2 Pattern structures and interval pattern structures . . . 10
2.3 Minimum description length principle . . . 12
2.4 Quality measures in pattern mining . . . 14
2.4.1 Pattern set quality measures in unsupervised settings . . . 15
2.4.2 Quality measures in supervised settings . . . 17

Chapter 3
Closure structure

3.1 Introduction . . . 24
3.2 Related work . . . 25
3.3 Basic notions . . . 26
3.4 Closure structure and its properties . . . 28
3.4.1 Keys and passkeys . . . 28
3.4.2 Closure structure . . . 29
3.4.3 Passkey-based order ideal . . . 30
3.4.4 Assessing data complexity . . . 30
3.4.5 Closeness under sampling . . . 31
3.5 The GDPM algorithm . . . 32
3.5.1 Computing the closure structure with GDPM . . . 32
3.5.2 The extent-based version of GDPM . . . 34
3.5.3 Complexity of GDPM . . . 35
3.5.4 Related approaches to key-based enumeration . . . 35
3.5.5 Computing passkeys. Towards polynomial complexity . . . 37
3.6 Experiments . . . 39

3.6.1 Characteristics of the datasets . . . 39
3.6.2 Computational performance . . . 40
3.6.3 Data topology or frequency distribution within levels . . . 42
3.6.4 Coverage and overlaps . . . 45
3.6.5 Usefulness of concepts . . . 45
3.6.6 Case study . . . 46
3.7 Discussion and conclusion . . . 49

Chapter 4
Pattern mining in binary data

4.1 Introduction . . . 54
4.1.1 Pattern types . . . 54
4.1.2 Exploring the pattern search space . . . 55
4.1.3 Interestingness measures . . . 56
4.2 Minimum description length principle in itemset mining . . . 57
4.3 Greedy strategy to reduce pattern search space . . . 63
4.3.1 Likely-occurring itemsets . . . 63
4.3.2 Likely-occurring itemsets and related notions . . . 66
4.3.3 Experiments . . . 67
4.4 Adapting the best practices of supervised learning to itemset mining . . . 73
4.4.1 KeepItSimple: an algorithm for discovering useful pattern sets . . . 73
4.4.2 Experiments . . . 78
4.5 Discussion and conclusion . . . 79

Chapter 5
Pattern mining in numerical data

5.1 Introduction . . . 83
5.2 Related work . . . 84
5.2.1 Numerical data preprocessing . . . 85
5.2.2 MDL-based approaches to pattern mining . . . 86
5.3 Basic notions . . . 87
5.3.1 Formalization of data and patterns . . . 87
5.3.2 Information theory and MDL . . . 88
5.3.3 Derivation of the plug-in codes . . . 90
5.3.4 ML-estimates of the parameters of the multinomial distribution . . . 90
5.4 Mint . . . 91
5.4.1 The model encoding . . . 91
5.4.2 The Mint algorithm . . . 93
5.5 Experiments . . . 97
5.5.1 Datasets . . . 97
5.6 Discussion and conclusion . . . 113

Chapter 6
Conclusion and future work

List of publications

Detailed summary in French

Bibliography

1

Introduction

1.1 Outline of pattern mining

Throughout human history, the way of discovering the world has been constantly evolving: starting from *empirical science*, that appeared thousands of years ago and consisted in describing the observations of natural phenomena, through *theoretical science*, that took shape last few hundred years and aimed at generalization of the observations by means of models, to *computational science*, that emerged last few decades and aimed at simulating complex phenomena. Nowadays, we may observe the fourth paradigm of extraction of knowledge about the world, *data science*, that combines theory, experiments, and simulation.

The rapid development of information technologies, wide availability of devices collecting different kinds of data, an increasing ease of access to low-cost computing power for processing huge amount of data, and growing synergy of interdisciplinary collaborations have made a shift in modern science and let us today speak of a new science paradigm: *data-intensive scientific discovery* (*data science* or *eScience*) [HTT09].

Although we started speaking of this new paradigm quite recently — roughly 10 years ago — the currently used process of knowledge extraction was described two decades before and is often referred to as "knowledge discovery in databases" (KDD) [Pia91]. Nowadays KDD exists within this new paradigm. In [FPS96] KDD is defined as "*the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns or relationships within a dataset in order to make important decisions*". According to the authors, the KDD process is interactive and includes five steps that are presented in Fig. 1.1.

Data mining is an important step of the KDD process. In [HKP11] it is defined as "*the automated or convenient extraction of patterns representing knowledge implicitly stored or captured in large databases, data warehouses, the Web, other massive information repositories, or data streams*". A specific field of data mining encapsulating unsupervised methods and dealing with patterns explicitly is called *pattern mining*. Patterns are usually understood as "repetitive fragments" in data [AH14].

However, any pattern mining approach *requires* the type of patterns — pattern language — to be set explicitly. Depending on data type and application, patterns can be represented by itemsets [FLV+17], hyper-rectangles in a multidimensional space [WDKG14, KKN11b], sequences [FVLK+17] (in particular, trajectories [BEJ+13], time series [Fu11], strings [FHK06], etc.), graphs [RKF12, KKVF14], fragment in images [FvL20], and so on. The type of patterns defines the *pattern search space* — the set of all possible patterns of a given type that are present in the dataset.

The choice of pattern type is closely related to the *pattern explosion* problem. Usually, pattern type is chosen in such a way that the corresponding pattern search space is rich enough to select patterns that fit the data but avoid pattern explosion. Moreover, in modern approaches, patterns are evaluated w.r.t. not only the data but also other patterns in order to ensure their non-redundancy [VVLS11, SV12]. As a consequence, the estimates of already considered interesting patterns are usually recomputed during pattern mining. Thus, from the computational point of view, discovering of a very limited subspace of potentially interesting patterns is more appropriate in practice than dealing with the whole pattern

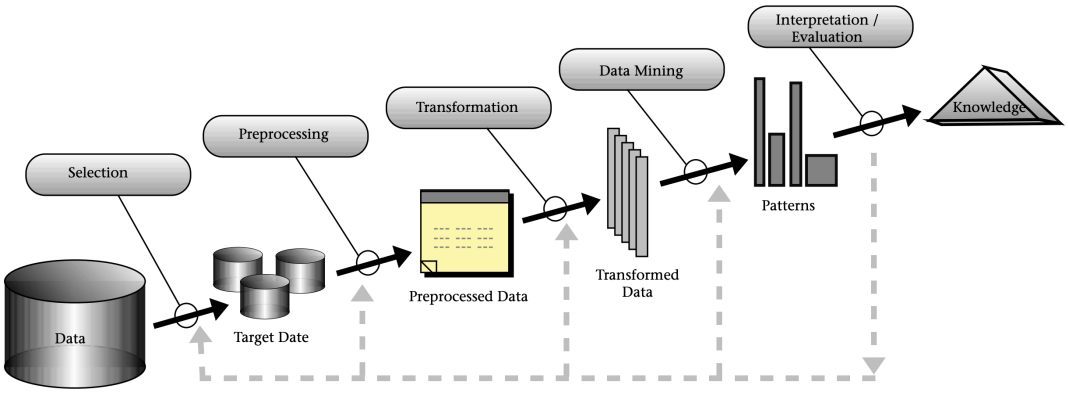


Figure 1.1: The steps of the KDD process [FPS96]

search space.

Another criterion used to choose pattern type is the ease of interpretation of the results, which means that patterns should allow for such a description that can be easily analyzed by experts.

Moreover, pattern mining implies that a measure of interest, or *interestingness measure*, is chosen in advance. The notion of *interestingness* is a key notion of pattern mining and the essential component of any pattern mining approach.

One of the major challenges of pattern mining is related to formalization of interestingness[AH14]. Since interestingness is subjective, development of a pattern mining method is accompanied by the following questions:

- What are the knowledge or intuition about data that should be exploited during pattern mining?
- What is the most appropriate way to incorporate this knowledge?
- How can data preprocessing help us to choose or establish the most appropriate interestingness measure?

Finally, the patterns are expected to be of good quality. We draw attention to the possible discrepancy between *interestingness measure* and *quality measure*. The interestingness measures evaluate "semantics" or "knowledge" lying behind patterns, while the quality measures usually formalize very intuitive expectations from the results, e.g., non-redundancy, diversity of patterns, and their comprehensiveness. In the scientific literature, the attempts to bridge the gap between these measures are related to the transition from *interestingness measures for patterns* to *interestingness measures for a pattern set*. With this regard, we emphasize the importance of paying sufficient attention to the compliance of pattern interestingness with some quality measures.

Evaluation of the quality of pattern sets presents an important challenge of pattern mining. A particularly acute question is how to select an appropriate pattern mining approach or to tune the parameters of a chosen approach when there is no ground truth or any other additional knowledge

available? Or, can we compare pattern mining methods that are based on different interestingness measures?

Pattern mining is a very broad field, and there are no universal solutions for the raised issues. In this study we make an attempt to address the aforementioned issues by specifying the problem.

We study binary and numerical data, where each object is represented by a tuple of either binary or numerical attributes. Even for seemingly simple data types the outlined issues are quite pronounced. We summarize the problems under study in the following questions:

- What can we *learn from data during preprocessing*, i.e., before articulating the pattern interestingness?
- How to *estimate the complexity* of data before launching a pattern miner?
- What kind of patterns we may use?
- How to *mitigate the pattern explosion problem*?
- What *techniques can be useful* even if no ground truth is available?
- How to mine a *good-quality* set of *interesting* patterns?
- What heuristics can we use to *improve the efficiency* of pattern miners?

This thesis is the description of our scientific journey of facing these challenges and unforeseen pitfalls to find elegant solutions written in a sound theoretical language.

Let us consider this work in detail.

1.2 Thesis structure and contribution

In this thesis we focus on datasets where objects are described by a tuple of binary or numerical attributes. This type of data is commonly referred to as *data tables*, *tabular data* or *data points in multidimensional space*.

In Chapter 2 we introduce the basic notions used in this study. We limit the type of patterns to *closed* ones. In Section 2.1 and 2.2, we introduce the notion of closeness for patterns in binary and numerical data within the framework of *formal concept analysis* (FCA) [GW99] and *pattern structures* [GK01], respectively. The latter is the generalization of FCA to the case of more complex data, e.g., sequences [JCNR18], partitions [CN14], graphs [Kuz99], etc. We use a specific type of pattern structures called *interval pattern structures* [KKN11b], that are meant to handle numerical datasets. Section 2.3 sketches some basics of the *minimum description length (MDL) principle* [Grü07], an information-theoretic principle for model selection that nowadays is widely used in pattern mining. And finally, in Section 2.4, we discuss the existing approaches to evaluation of the results of pattern mining.

The main contribution of the thesis is presented in Chapter 3, 4, and 5.

In Chapter 3 we focus on binary data and the problem of uncovering its underlying *intrinsic structure* underlying the binary data. This chapter sheds the light on the role and particular usefulness of closed itemsets. Using the solid mathematical foundations of FCA, we formalize this intrinsic structure underlying the data by considering so-called *equivalence classes* of patterns. We introduce specific elements — *passkeys* — to characterize subsets of "equivalent" itemsets and use the passkeys to define the *closure structure*. This chapter contains *important theoretical results*:

- The formal model for estimating *itemset and data complexity* is introduced. Thus, by analogy with the algorithmic complexity or the Vapnik-Chervonenkis dimension of a classification model [VC15], we introduce formal tools to estimate the complexity of both itemsets and the corresponding binary datasets. Moreover, the introduced formalism allows to deal with the complexity of itemsets and the corresponding datasets based on the common ground.
- The results on closeness and complexity of itemsets (and datasets) under *sampling* are obtained.

- The GDPM algorithm for gradual discovery of the closure structure is proposed. This algorithm computes the closure structure by levels w.r.t. the complexity of itemsets associated with them. GDPM has polynomial delay.
- It is proven that knowing the complexity of an itemset (the closure level it belongs to) allows for decreasing the complexity of a #·coNP-complete problem of counting passkey to polynomial under condition that itemset complexity is a constant.
- The number of passkeys for an equivalence class does not exceed the number of keys, and we show that the number of keys can be exponentially larger than the number of passkeys.

The chapter contains some *practical contributions* as well, including:

- The introduced closure structure provides a generic representation of "data topology" w.r.t. a measure of interest.
- The closure structure allows for a succinct representation of implications, providing information on the amount of non-redundant[1] implications with the antecedent and consequent of a given size can be found in the dataset.
- Given the closure structure or its first levels, we may understand how itemsets are distributed by levels (w.r.t. their complexity) and when to stop the search for interesting itemsets with a minimal "loss".

To sum up, Chapter 3 is focused on the structural representation of binary data, some theoretical aspects of data complexity and the related computational issues. The main practical impact of the closure structure is that it allows for simple and generic representation "data topology" w.r.t. a chosen measure.

In Chapter 4 we continue with *pattern mining in binary data*, but driven by a different motivation. In fact, the closure structure introduced in Chapter 3 helps to understand how data is structured, while the methods of pattern mining are meant to describe the knowledge hidden in data. Thus, we shift the focus from "*form*" of data to its "*content*". In the methods of Chapter 4 the primacy is given to heuristics. Firstly, in Section 4.1, we provide a short introduction to pattern mining in binary data: (i) we give a summary on different types of patterns that can be defined in binary data, and discuss why the closed itemsets is a good choice, then (ii) we consider various approaches to itemset search space exploration, and finally, (iii) we describe the common approaches to evaluate interestingness of itemsets, explain the difference between mining of *a set of patterns* and *a pattern set*. We also show how MDL can be applied to itemset mining.

Then, we proceed to the main results related to itemset mining. The contribution of this chapter is two-fold: we propose (i) a general-purpose greedy strategy for exploration of an itemset search subspace, and (ii) a greedy approach to itemset pattern mining.

The first contribution is presented in Section 4.3. We focus on the problem of gradual discovery of itemset search space. We introduce so-called *likely-occurring (closed) itemsets* as well as an algorithm to compute them. The proposed approach is particularly useful when the exhaustive enumeration of all or frequent (closed) itemsets is hard, or frequency is not an appropriate criterion to restrict the itemset search space. The method uses a greedy strategy but relies on a theoretical basis. This work has a *practical impact*, as we demonstrate its usefulness of the proposed method in association rule mining.

We also point to its shortcomings: if there is a discovery method for itemset search space that is tailored to a chosen interestingness measure, it is preferable to use this method rather than the proposed one.

The results of this work were presented at the 15th *International Conference on Formal Concept Analysis*, ICFCA-2021 [MKN19b].

The second contribution of this chapter is presented in Section 4.4. We address two related problems: (i) the conformity of an interestingness measure with intuitive quality measures for itemset sets, (ii) the lack of the ground truth needed to perform a formal evaluation of the results of itemset mining. We consider the model of supervised learning that is the closest to itemset mining — decision tree ensembles

[1] The "non-redundancy" of association rules is discussed in Chapter 3

— and adapt its best heuristics under the assumption that if they work well in supervised settings they may work well in unsupervised settings. With this regard, the *contribution* of this part can be summarized as follows:

- We propose an itemset mining approach for computing a *multi-model description* of data by means of *biclosed* itemsets (i.e., "low-complexity" itemsets), and modify a model introduced in [VVLS11] to improve the quality of the results. In experiments, we show that the proposed method allows for getting itemset sets that have better quality w.r.t. intuitive quality measures.

The results of this work have been presented at the 9th *European Starting AI Researchers' Symposium* STAIRS@ECAI-2020 [MKN20].

Chapter 5 is devoted to mining patterns in numerical data. We restrict the pattern search space to hyper-rectangles, which are essentially interval pattern concepts introduced in Section 2.2. The main impact of this chapter is the following:

- We propose an MDL-based approach to numerical pattern set mining, called Mint, which explores pattern search space gradually and does not suffer from exponential pattern explosion.
- In experiments we show that Mint is efficient in practice, it allows for computing non-redundant set of informative patterns, and provides better results than its MDL-based competitors RealKrimp [WDKG14] and Slim [SV12].

The chapters containing the main results, namely Chapter 3, 4, and 5, are focused on diverse aspects of pattern mining and related tasks and can be read independently.

Chapter 6 wraps up the work. We discuss the main results and directions of future work.

2

Background

Contents

2.1 Formal concept analysis **7**
2.2 Pattern structures and interval pattern structures **10**
2.3 Minimum description length principle **12**
2.4 Quality measures in pattern mining **14**
2.4.1 Pattern set quality measures in unsupervised settings 15
2.4.2 Quality measures in supervised settings 17

In this chapter we describe the basic concepts used throughout the thesis. Firstly, in Section 2.1, we introduce the fundamental notions related to pattern mining in binary data within the framework of the formal concept analysis (FCA). This framework provides a very generic formalism for dealing with itemsets, tiles, and other types of patterns in binary data. Moreover, it allows for structuring the pattern search space into classes of "equivalent patterns".

The strength of the FCA formalism is that it not only provides uniform tools for dealing with a large spectrum of patterns in binary data but also can be easily generalized to more complex data, in particular to numerical data. In Section 2.2 we present interval pattern structures, which is a generalization of FCA to the case of numerical data. In Chapter 5 we propose an approach for mining a set of interval pattern concepts.

After introducing the types of binary and numerical patterns we use in this study, we introduce in Section 2.3 the basics of the minimum description length principle, which we use for mining binary and numeric patterns.

Finally, in Section 2.4, we give a quick overview of the existing approaches to evaluate the results of pattern mining. We discuss only very general notions, some specific quality measures will be introduced in the sections where we use them.

2.1 Formal concept analysis

Formal concept analysis [GW99] is a branch of applied lattice theory, where, starting from the binary relation between objects and attributes, one builds a set of partially ordered *formal concepts*, i.e., pairs composed of a set of objects and the set of attributes shared by these objects.

FCA has been applied to a wide range of tasks of knowledge discovery and data mining [PIKD13]. In particular, FCA provides strong mathematical foundations for itemset mining. The approaches initially proposed to enumerate of formal concepts in FCA are then widely used for enumeration of frequent closed itemsets in data mining [KO02]. However, FCA is hardly ever considered in the data mining community as a theoretical framework for pattern mining, and advances of FCA rarely find immediately a wide application in data mining. For example, LCM [UAUA03, UAUA04] — one of the most efficient algorithm for enumeration of frequent closed itemsets — was developed independently from the

CbO algorithm [Kuz93] and can be considered as its improved version polished with efficient implementation solutions. Most notably, CbO was proposed 10 years earlier than LCM. Moreover, within the FCA framework a lot of approaches for selection of interesting concepts (closed itemsets) have been proposed [KM18]. FCA accommodates a lot of other valuable contributions to pattern mining [PKID13a]. In particular, in [BKN15b] the authors propose an interestingness measure, called Δ-measure, that exhibits anti-monotonicity and, thus, can be largely used along with frequency to discover gradually itemset search space. Furthermore, the algorithm for itemset discovery based on Δ-measure has incremental polynomial delay, whereas frequency-based algorithms do not provide any guarantees.

In addition, FCA lays the groundwork for association rule and implication mining [PBTL99a, PBTL99b, LS05]. The question of the efficient enumeration of non-redundant association rules and implications has been exhaustively studied in FCA both from theoretical and practical standpoints. We discuss it in detail in Section 3.2.

The structure formed by partially ordered formal concepts — concept lattice — is commonly used to build taxonomies and ontologies in different domains [PKID13b].

FCA is also used to build classification models. Some well-known methods and notions of machine learning can be expressed in terms of FCA, e.g., the version space learning [Mit79, GK03], decision trees [Kuz04], JSM-method [Kuz89, Kuz91].

We limit the review of the existing directions of FCA to this short summary. Interested readers are invited to read an exhaustive description of main research topics in the FCA community in [PIKD13]. Let us proceed to the main definitions.

Definition 1. *A* formal context *[GW99] is a triple $\mathbb{K} = (G, M, I)$, where G is a set of objects, M is a set of attributes and $I \subseteq G \times M$ is a relation called* incidence relation*, i.e., $(g, m) \in I$ if object g has attribute m.*

The derivation operators $(\cdot)'$ are defined for $A \subseteq G$ and $B \subseteq M$ as follows:

$$A' := \{m \in M \mid \forall g \in A : gIm\}\,,\ B' := \{g \in G \mid \forall m \in B : gIm\}\,.$$

A' is the set of attributes common to all objects of A and B' is the set of objects sharing all attributes of B. The double application of $(\cdot)'$ is called closure operator and denoted by $(\cdot)''$.

The closure operator is extensive, idempotent and monotone, i.e., for any $A_1, A_2 \subseteq G$ the following holds:

- $A_1 \subseteq A_1''$ (extensive);
- $(A_1'')'' = A_1''$ (idempotent);
- $A_1 \subseteq A_2 \to A_1'' \subseteq A_2''$ (monotone).

Sets $A \subseteq G$, $B \subseteq M$, such that $A = A''$ and $B = B''$, are said to be *closed*.

Definition 2. *For $A \subseteq G$, $B \subseteq M$, a pair (A, B) such that $A' = B$ and $B' = A$, is called a* formal concept*, A and B are called the* extent *and the* intent*, respectively.*

In data mining [PBTL99c], an attribute is also called an item, an *intent* is called a *closed itemset* or a *closed pattern*, and the extent is also called the *image* of the itemset.

A partial order $\leq$ is defined on the set of concepts as follows: $(A, B) \leq (C, D)$ iff $A \subseteq C\,(D \subseteq B)$, a pair (A, B) is a subconcept of (C, D), while (C, D) is a superconcept of (A, B). With respect to this partial order, the set of all formal concepts forms a lattice $\mathcal{L}$ called the *concept lattice* of the formal context (G, M, I). The size of the concept lattice is bounded from above by $O(2^{\min(|G|,|M|)})$.

Definition 3. *For an itemset X its* equivalence class *$Equiv(X)$ is a set of itemsets with the same extent, i.e., $Equiv(X) = \{Y \mid Y \subseteq X'', X' = Y'\}$.*

Thus, a *closed itemset* X'' is the maximal itemset in the equivalence class $Equiv(X)$. The concept lattice represents concisely the whole itemset search space, i.e., all possible combinations of items, where itemsets having the same support are encapsulated in one equivalence class represented by the closed itemset. An equivalence class is called *trivial* if it consists of only a closed itemset.

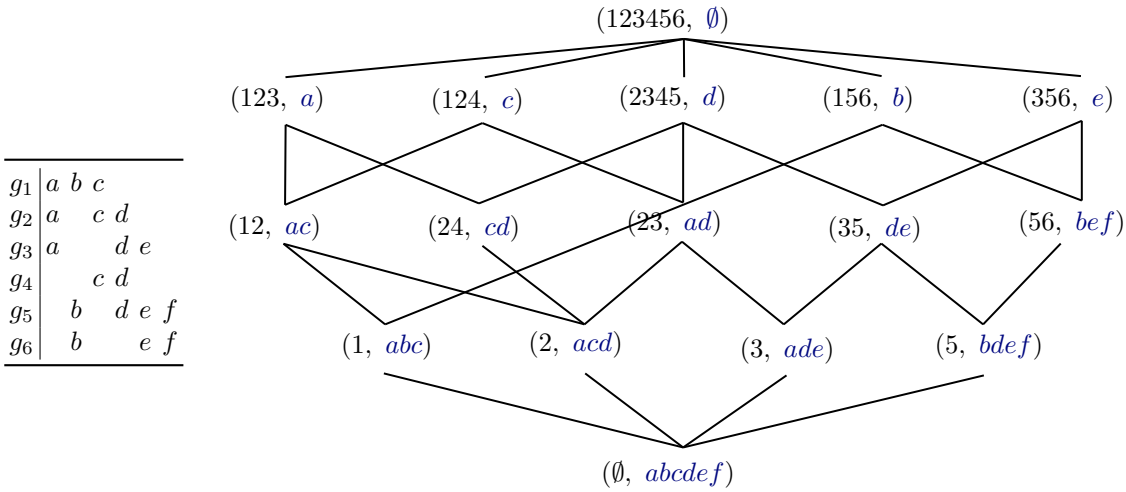


Figure 2.1: A binary dataset, or formal context, (left) and its concept lattice (right)

Definition 4. *Let B be a closed itemset. Then any member of the equivalence class $Y \in Equiv(B)$ is a* generator *of B.*

Definition 5. *A* key *$X \in Equiv(B)$ is a* minimal generator *in $Equiv(B)$ if for every $Y \subset X$ it holds that $Y' \neq X' = B'$.*

Minimal generators are the generators that are minimal in the equivalence class by inclusion. More formally, a generator X is minimal if there is no generator $Y \subset X$ in the equivalence class $Equiv(X)$ that is included in X.

Example. Let us consider a formal context (G, M, I) from Fig. 2.1 (left). The corresponding lattice is given on the right. For the sake of simplicity, we replace each "cross" corresponding to gIm in the formal context with the attribute m, and represent sets as strings, e.g., we write "abc" for "$\{a, b, c\}$" and "123" for "$\{g_1, g_2, g_3\}$". Among all formal concepts there are five concepts that have non-trivial equivalence classes, i.e., where there exists at least one itemset $X \subset A$ such that $X'' = A$. All non-trivial equivalence classes and the corresponding generators are listed in Table 2.1. The set of minimal generators may differ from the set of generators. For instance, the closed itemset bef has two minimal generators, namely f and be, since there is no generator in $Equiv(bef)$ that is entirely included in f or be. The other 3 generators of bef contain either f, or be, or both of them.

Table 2.1: Closed itemsets and the corresponding minimal generators and equivalence classes, omitting cases with trivial equivalence classes, for the dataset from Fig. 2.1. Closed itemsets are given in column B

B	minimal generators	$Equiv(B)$ (generators)
ade	ae	ae, ade
abc	ab, bc	ab, bc, abc
bef	f, be	f, be, ef, bf, bef
$bdef$	bd, df	bd, df, def, bdf, bde, $bdef$
$abcdef$	af, ce, cf, abd, abe, bcd	af, ce, cf, abd, abe, abf, ace, acf, adf, aef, bcd, bce, bcf, bde, bdf, cde, cdf, cef, def, ..., $abcdef$*

* The equivalence class includes all itemsets that contain a minimal generator of $abcdef$.

In this study we rely on closed itemsets rather than itemsets in general since a closed itemset provides a *lossless representation* of these itemsets and a concise representation of implications and association rules [PBTL99c].

2.2 Pattern structures and interval pattern structures

FCA provides a mathematically sound framework for dealing with binary data. However, to deal with more complex data within FCA one needs to binarize data. The choice of an appropriate binarization can be very difficult and often accompanied by loss of information. Pattern structures were proposed to tackle this issue and to deal directly with complex data in compliance with the FCA theory [GK01].

Definition 6. *Let G be some set, $(D, \sqcap)$ be a meet-semilattice*[2] *and $\delta : G \to D$ be a mapping. Then a* pattern structure *[GK01] is a triple $\mathbb{P} = (G, (D, \sqcap), \delta)$ provided that the set $\delta(G) = \{\delta(g) \mid g \in G\}$ generates a complete subsemilattice $(D_\delta, \sqcap)$ of $(D, \sqcap)$.*

To ensure that the subsemilattice $(D_\delta, \sqcap)$ is complete it is sufficient that one of the following conditions be satisfied: (i) $(D, \sqcap)$ is complete, or (ii) G is finite.

For a pattern structure $(G, (D, \sqcap), \delta)$ the derivation operators, denoted $^\diamond$,that make a Galois connection, are defined as follows:

$$A^\diamond := \prod_{g \in A} \delta(g) \text{ for } A \subseteq G, \; d^\diamond := \{g \in G \mid d \sqsubseteq \delta(g)\} \text{ for } d \in D.$$

$A^\diamond$ is a common description to all objects in A, while $d^\diamond$ is a set of objects whose descriptions subsume d. The (subsumption) order on the descriptions is given by $c \sqsubseteq d \Leftrightarrow c \sqcap d = c$

Definition 7. *A* pattern concept *of a pattern structure $(G, (D, \sqcap), \delta)$ is a pair (A, d), where $A \subseteq G$ and $d \in D$ such that $A^\diamond = d$ and $d^\diamond = A$. A is called the* pattern extent *and d is called the* pattern intent.

Pattern structures can be defined on a wide variety of data types, e.g., sequences [CBK+17], partitions [BKN14a, CN14], convex polygons [BKRK17], graphs [KS05], parse thickets (i.e., syntactic trees augmented with arcs reflecting inter-sentence relations) [SGIK14], etc.

In this study we use a special type of pattern structures called *interval pattern structures* [KKN11b] that were introduced to deal with numerical data. The pattern concepts in the interval pattern structures are called *interval pattern concepts*. Simply speaking, the interval pattern concepts are hyper-rectangles in multidimensional space. In Chapter 5 we propose a pattern mining approach where we use the hyper-rectangles as patterns.

Within the formalism of pattern structures, we may represent numerical data as a many-valued context $\mathbb{K}_{num} = (G, S, W, I_{num})$, where $(g, s, w) \in I_{num}$ means that object g has value w for attribute s.

Considering a dataset over m numerical attributes a pattern intent is a tuple of $m = |S|$ intervals, i.e. ranges of numerical values defined by a lower bound and an upper bound denoted $[l, u]$. Let $c = \langle [a_i, b_i] \rangle_{i \in \{1,\ldots m\}}$ and $d = \langle [e_i, f_i] \rangle_{i \in \{1,\ldots m\}}$ be two interval pattern intents. For c and d the similarity (*meet*) operator is defined as follows: $c \sqcap d = \langle min(a_i, e_i), max(b_i, f_i) \rangle_{i \in \{1,\ldots m\}}$. The subsumption order therefore is defined by $c \sqsubseteq d \Leftrightarrow [e_i, f_i] \subseteq [a_i, b_i], \forall i \in \{1, \ldots, m\}$.

Example. An example of numerical context is given in Fig. 2.2 on the left, the corresponding pattern structure is given on the right. Broadly speaking, given a set of objects that constitute the extent of a pattern, the pattern intent is the smallest rectangle that envelop the pattern extent. For example, the smallest rectangle that envelops $\{g_1, g_3\}$ is $[1, 2] \times [3, 4]$. This rectangle includes object g_2 as well, thus the pattern extent is $\{g_1, g_2, g_3\}$, and the interval pattern concept is given by $(\{g_1, g_2, g_3\}, \langle [1, 2], [3, 4] \rangle)$.

The meet $\sqcap$ of two interval pattern intents is the smallest rectangle that envelops both of them. Let us consider two interval pattern concepts $(\{g_1, g_2, g_3\}, \langle [1, 2], [3, 4] \rangle)$ and $(\{g_1, g_4\}, \langle [1, 4], [3, 3] \rangle)$. Then, the meet of their intents is $\langle [1, 2], [3, 4] \rangle \sqcap \langle [1, 4], [3, 3] \rangle = \langle [1, 4], [3, 4] \rangle$.

A many-valued context can be transformed into a binary context in several ways, some of which do not entail any loss of information (at the cost of an explosion of the number of attributes). The lossless data transformation is called *interordinal scaling* (IS) [GW99]. The interordinal scaling consists in replacing each value v of attribute m by two binary attributes: "$m \geq v$" and "$m \leq v$". An interordinally scaled numerical context from Fig. 2.2 is given in Fig. 2.3. This formal context contains a lot of redundant attributes, e.g., "$m_1 \geq 1$", "$m_1 \leq 4$", "$m_2 \geq 3$" and "$m_2 \leq 4$". In [KKN11b] the authors point out not only the local redundancy of IS-itemsets (i.e., there may exist several arbitrary IS-itemsets that

[2] A partially ordered set which has a meet (greatest lower bound) for any nonempty finite subset.

	m_1	m_2
g_1	1	3
g_2	1	4
g_3	2	4
g_4	4	3

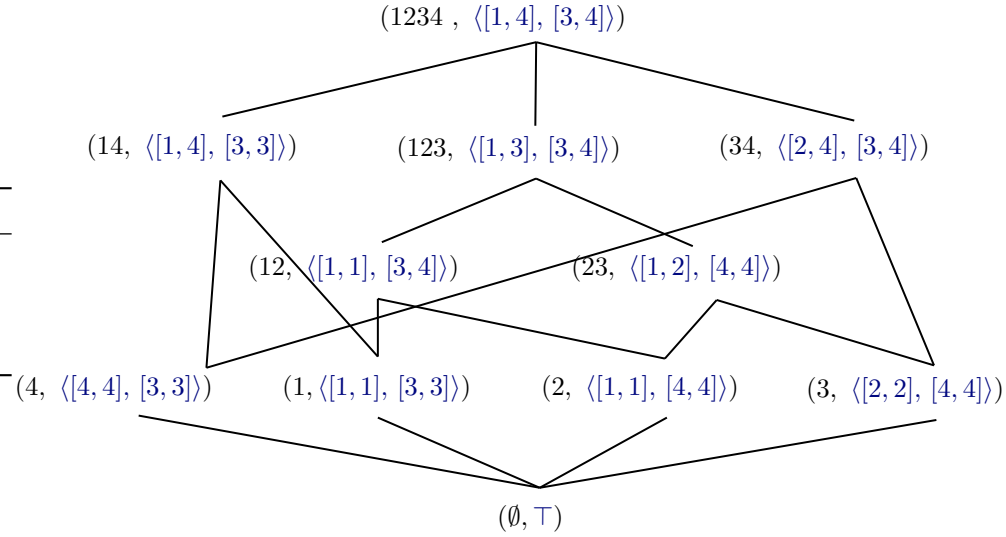


Figure 2.2: Numerical context (left) and the corresponding interval pattern structure (right)

	≤ 1 m_1	≥ 1 m_1	≤ 2 m_1	≥ 2 m_1	≤ 4 m_1	≥ 4 m_1	≤ 3 m_2	≥ 3 m_2	≤ 4 m_2	≥ 4 m_2
g_1	×	×	×		×		×	×	×	
g_2	×	×	×		×			×	×	×
g_3		×	×	×	×			×	×	×
g_4		×		×	×	×	×	×	×	

	a	b	c	d	e	f	g
g_1	×	×	×			×	
g_2	×	×	×				×
g_3		×	×	×			×
g_4		×		×	×	×	

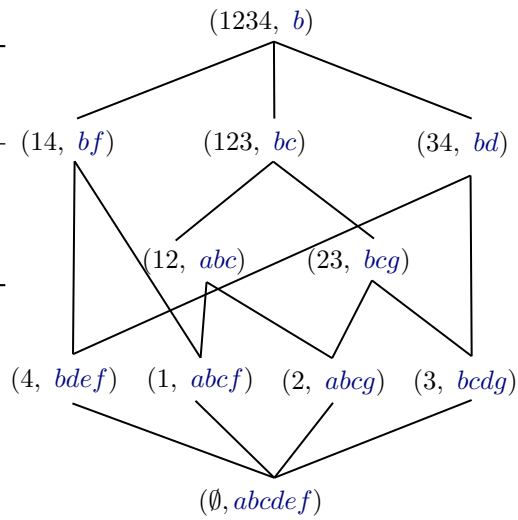


Figure 2.3: Interordinal-scale context (upper left), reduced context (bottom left) and the corresponding concept lattice (right)

correspond to a single interval pattern intent) but also global (i.e., the number of IS-generators is not less than the number of generators of interval patterns).

The reduced context is given in Fig. 2.3 below the IS formal context. We removed the attributes with the repetitive extents (i.e., the repetitive columns of the formal context). However, this context is not minimal, meaning that removing the attributes "b" and "e" does not change the lattice structure.

In practice, especially for real-valued data, one usually *scales* data and applies *one-hot encoding*. Scaling consists in defining a partition, i.e., choosing a limited number of values as cut-points for each attribute rather than using all values that occur in the data, while one-hot encoding consists in replacing a particular value with the interval containing this value. We give an example of scaling and one-hot encoding in Fig. 2.4. There are a lot of techniques to define the partition (i.e., to discretize data), we discuss them in detail in Section 5.2.1.

Thus, interval pattern structures generalize the concepts from FCA from binary data to numerical

	s_1	s_2
g_1	[1,4)	[3,4)
g_2	[1,4)	[4,5)
g_3	[1,4)	[4,5)
g_4	[4,5)	[3,4)

	$s_1 \in [1,2)$	$s_1 \in [1,4)$	$s_2 \in [3,4)$	$s_2 \in [4,5)$
	a	b	c	d
g_1	×		×	
g_2	×			×
g_3	×			×
g_4		×	×	

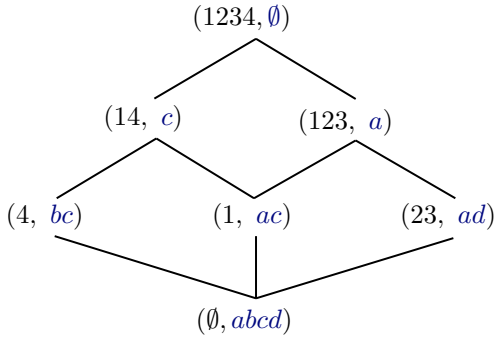


Figure 2.4: The scaled version of the context from Fig. 2.2 (left), its one-hot encoded equivalent (middle), and the corresponding formal concept lattice

data. In Chapter 5 we discuss the problem of pattern mining in numerical data in detail.

2.3 Minimum description length principle

The minimum description length principle (MDL) is a principle that is widely used in data mining and knowledge discovery for model selection [Gal20]. Often, the methods based on MDL are referred to as "learning by compression", since "learning" in these approaches is equated to "finding regularities", and the more regularities are found, the more the data can be compressed.

MDL is originated from the notion of the Kolmogorov complexity [VL97]. The *Kolmogorov complexity* of a data sequence is defined as the length of the shortest program that prints this sequence and then *halts*. The basic concept related to the Kolmogorov complexity is a *computing machine* (or, in more practical terms, "programming language"). In [Sol64] Solomonoff described the concept of a "*universal machine*" M_U as a subclass of the universal Turing machine such that for any arbitrary computing machine M and any string x there exists a string α, as a function of M_U and M, such that $M_U(\widehat{\alpha x}) = M(x)$, where $M(x)$ is the output of M given input x, and $\widehat{\alpha x}$ is the concatenation of strings α and x. Thereby α can be viewed as a translation instruction.

In the same work, Solomonoff proved that asymptotically for any two universal machines M_1 and M_2, the difference between the shortest code lengths for the same input sequence is finite and depends only on M_1 and M_2. The latter means that any program of one universal machine can be translated to the equivalent program for another universal machine with a program of a finite length. This result was obtained independently by Solomonoff [Sol64], Kolmogorov [Kol65], and Chiatin [Cha69], and now is known as the *invariance theorem*.

However, the Kolmogorov complexity is known to be uncomputable in general, and in practice, the length of the translating program α (mentioned in the invariance theorem above) can be too large w.r.t. the length of the data sequence x. To circumvent this issue, in practice one usually resorts to a simplified version of this problem: one uses a restricted class of description methods (or "machines") such that the length of the shortest description (or "program") be computable. For the sake of uniformity, further, we use the notion of "model" for a description method.

Hence, in practical MDL we deal with a restricted class of models $\mathcal{M}$. One distinguishes crude and refined versions of MDL.

The crude version is formulated as follows: given a class of models $\mathcal{M}$, select one model $M \in \mathcal{M}$ that minimizes the total length, i.e., $M = \arg\min_{M \in \mathcal{M}}(L(M) + L(D|M))$, where $L(M)$ is the length, in bits, of the description of the model; and $L(D|M)$ is the length, in bits, of the description of the data when encoded by this model.

Multiple pattern mining methods are based on the crude version of MDL since it allows for selecting a specific model, i.e., a pattern set. The crude version of MDL is used to encode various pattern types, e.g., sequences [CAP18], graphs [BCF20], rules [PvL20], etc.

In contrast to the crude version of MDL, the refined one is used encoded with the entire model family

Refined MDL is based on a one-part code length $\overline{L}(D|\mathcal{M})$. This code is designed in such a way that whenever$M \in \mathcal{M}$ ensures small $L(D|M)$ then the code length $\overline{L}(D|\mathcal{M})$ will also be small [Grü07]. The class of models $\mathcal{M}$ is usually parametric, and, in the literature, $\mathcal{M}$ may be referred to as a single *full model*, where M corresponds to some values for the parameters of this model. In this work, for the sake of uniformity, we call $\mathcal{M}$ a *class* of models both for crude and refined MDL.

In the refined MDL, $\overline{L}(D|\mathcal{M})$ is called the *stochastic complexity* of the data given the class of models. Apart from the stochastic complexity, another fundamental notion of the refined MDL is *parametric complexity* $COMP(\mathcal{M})$. It characterizes the "richness" of the model class $\mathcal{M}$. The both complexities are related as follows: $\overline{L}(D|\mathcal{M}) = L(D|\hat{M}) + COMP(\mathcal{M})$, where $\hat{M}$ denotes the distribution in $\mathcal{M}$ which minimizes the length of data $L(D|\hat{M})$. Thus, the stochastic complexity is decomposed into the terms characterizing goodness-of-fit and complexity of the model class.

The goal of MDL is to select a model, and it uses *compression* as a measure of model quality. MDL requires compression to be lossless, i.e., the encoded data sequence is guaranteed to be decoded without any distortion (loss of information). Thus, the application of the MDL principle implies encoding and decoding that can be summarized in the following communication model: given a sequence that should be sent from a transmitter to a receiver, the transmitter, instead of encoding each symbol uniformly, replaces repetitive sub-sequences with code words. Hence, instead of a symbol-wise encoded sequence, the transmitter sends a sequence of code words and a dictionary. The dictionary contains all used code words and the associated sub-sequences. Using the dictionary, the receiver is able to reconstruct the original sequence. The MDL principle is applied to decide which sub-sequences should be replaced with code words and which code words should be chosen for these sub-sequences.

To make the encoded sequence decodable, one may use either the *uniform* (fixed-length) or *prefix-free* codes. The *prefix code* is a code system where the codes have variable lengths and where there is no whole code word that is a prefix of any other code, thereby ensuring that the code is decodable. Allocating shorter code words to strings that appear often in the message allows to compress it.

One of the most famous prefix codes are Shannon-Fano [Sha48, Fan49] and Huffman [Huf52], the other codes can be found in [CT91]. However in MDL one is interested not in actual code words but rather in their lengths. Thus, for the length of x one commonly uses

$$L(x) = \lceil \log_2 P(x) \rceil, \tag{2.1}$$

referred to as Shannon-Fano codes. $\lceil \cdot \rceil$ is the ceiling function. Further, we write log for $\log_2$ and put $0 \log 0 = 0$.

The probability is usually estimated by the usage (occurrence) of x in the encoded sequence, i.e.,

$$P(x) = \frac{usage(x)}{\sum_{x \in \mathcal{A}} usage(x)}.$$

Dealing with the prefix codes, it is important to make sure that they satisfy *Kraft-McMillan inequality* [Kra49, McM56]. The Kraft-McMillan inequality gives a necessary and sufficient condition for the existence of a prefix code. The inequality has the following form: $\sum_{x \in \mathcal{A}} 2^{L_C(x)} \leq 1$, where $\mathcal{A}$ is the alphabet (or a set of fragments that should be encoded), $L_C(x)$ is the length of the code word associated with $x \in \mathcal{A}$.

The Shannon-Fano codes (Equation 2.1) have several drawbacks, which we discuss in detail in Section 5.3.2.

Uniform (fixed-length) codes are also widely applied in MDL. Let $\mathcal{A}$ be a finite alphabet. In case, where neither transmitter nor receiver do not know which symbols are more likely than the others, but still are interested in minimization of the number of bits needed to send the message, the symbols can be encoded with the fixed-length codes. The minimal length of the code word x then is given by $L_U(x) = \lceil \log |\mathcal{A}| \rceil$. The uniform codes are known to satisfy the *minimax property*, i.e., ensure the minimal number of bits needed to send a given message in the worst case. More formally, $\max_{x \in \mathcal{A}} L_U(x) = \min_{L \in \mathcal{L}_\mathcal{A}} \max_{x \in \mathcal{A}} L(x)$, where $\mathcal{L}_\mathcal{A}$ denotes the set of all length functions for prefix description methods over $\mathcal{A}$.

In the case, when the alphabet is an infinite countable set, the so-called "quasi-uniform description" method can be used to construct the codes close to uniform. More formally, let $\mathcal{A}_1 \subset \mathcal{A}_2 \subset ...$ such

that $\mathcal{A} = \bigcup_{\gamma \in \mathbb{N}} \mathcal{A}_\gamma$ be a family of sets. The goal is to define a coding that for every $x \in \mathcal{A}$ returns the code word of x of the length close to $\lceil \log |\mathcal{A}_\gamma| \rceil$, where γ is the smallest γ such that $x \in \mathcal{A}_\gamma$. Thus, for $x \in \mathcal{A}_\gamma \setminus \mathcal{A}_{\gamma-1}$ the length of the code should be close to $\lceil \log |\mathcal{A}_\gamma| \rceil$, for $x \in \mathcal{A}_{\gamma-1} \setminus \mathcal{A}_{\gamma-2}$ the length should be close to $\lceil \log |\mathcal{A}_{\gamma-1}| \rceil$, and so on. A more detailed description can be found in [Grü07].

This "quasi-uniform description" method is commonly used to encode integers, when the upper bound is not known, in MDL-based models. In this work, we use the quasi-uniform length described in [Ris83], where the length, in bits, of natural number $n \in \mathbb{N}$ is given by $L_{\mathbb{N}}(n) = \log n + \log \log n + \log \log \log n + \ldots + \log c_0$. The summation stops at the first negative term. The value $c_0 \approx 2.865$ is chosen so that the Kraft-McMillan inequality is satisfied with equality: $\sum_{j>1} 2^{-L_{\mathbb{N}}(j)} = 1$. The corresponding distribution with lengths $P_{\mathbb{N}}(j) = 2^{-L_{\mathbb{N}}(j)}$ is known as the *universal prior for the integers*. $L_{\mathbb{N}}(n)$ increases extremely slowly with n.

In this work, we use the crude MDL and consider it applying to itemset mining in Section 4.2 and to mining hyper-rectangles in numerical data in Section 5.4.

2.4 Quality measures in pattern mining

As mentioned above, the main difficulty in the evaluation of the results of pattern mining is the lack of ground truth. Put differently, the actual patterns are unknown. In some cases, when several miners use the same objective, the evaluation process can be performed by considering the values of the chosen interestingness measure. For example, in MDL-based methods, the compression ratio may be chosen as a quality measure. However, an interestingness measure does not always reflect fully the common intuition on the quality results.

A straightforward method to mitigate this issue is to apply quality measures that reflect the desired properties. We consider the quality measures within the following categorizations:

- measures for a set of *patterns* and for a *pattern set*;
- measures for evaluating patterns in *unsupervised* and *supervised* settings.

The key difference between measures for a set of patterns and a pattern set is that the first measures are applied to each pattern individually and then aggregated across the set, without taking into account overlaps and interactions between the patterns. Thus, the quality of the set of patterns usually is an aggregated value of a certain quality measure, while measures for pattern sets are applied to the whole pattern set, thus patterns are evaluated w.r.t. other patterns.

The difference between methods of the second group is that the "supervised" measures are applied in the case where the ground truth is known, while for the "unsupervised" measures no additional information is required.

Usually, the quality measures formalize some intuition of the analysts about interestingness. For example, such characteristics as diversity, coverage, meaningfulness, etc. In [KH06b] the authors propose the following criteria of interestingness:

- no two patterns should cover (approximately) the same set of examples;
- no pattern should cover (approximately) the complement of another pattern;
- no pattern should cover (approximately) a logical combination of two or more other patterns;
- patterns should be (approximately) mutually exclusive;
- when using patterns as input to a classifier, the pattern set should lead to the best performing classifier;
- patterns should lie on the convex hull of all patterns in ROC-space.

In this section we do not attempt to provide an exhaustive list of possible formalizations but rather provide quite intuitive and commonly used quality measures.

g_1	a	b	c				
g_2	a	b	c				
g_3	a	b					
g_4	a	b					
g_5	a	b				f	g
g_6				d	e	f	g
g_7				d	e	f	g

abc	ab	$abfg$	$defg$	fg	
1	1	0	0	0	$c_1 = 11000$
1	1	0	0	0	
0	1	0	0	0	$c_2 = 01000$
0	1	0	0	0	
0	1	1	0	1	$c_3 = 01101$
0	0	0	1	1	$c_4 = 00011$
0	0	0	1	1	

Figure 2.5: Dataset (left) and its shattering matrix (right) computed on pattern set $\{abc, ab, abfg, defg, fg\}$

2.4.1 Pattern set quality measures in unsupervised settings

Below we consider the most common quality measures applicable in unsupervised settings.

Joint entropy. *Joint entropy* was proposed in [KH06a] to evaluate the diversity of a pattern set. It is applicable to any kind of patterns since each pattern can be represented as a binary attribute, where 1's at i-th position means that the pattern describe object g_i. We say that pattern p describes object g_i if p is included in the description of g_i (if p is an itemset) or p contains the point corresponding to g_i (if p is a hyper-rectangle and g_i is a point in a multidimensional space).

Let $\mathcal{P} = \{p_1, \ldots, p_k\}$ be a set of patterns, and $G = \{g_1, \ldots, g_n\}$ be a set of objects described in the analysed dataset.

Definition 8. *The* signature *of a pattern p on G is called a vector $s \in \{0,1\}^n$ whose i-th element is equal to 1 if p describes object g_i, and 0 otherwise.*

The signatures of the patterns combined together (as columns) make a $|G| \times |\mathcal{P}|$-matrix that we call "shattering matrix". The obtained matrix may be considered in the context of the Vapnik-Chervonenkis theory. Then the "richness" of the pattern set $\mathcal{P}$ is related to VC-dimension [HTF09]. Put differently, we can measure how many groups of objects have unique descriptions in the space of the signatures of patterns from $\mathcal{P}$.

Definition 9. *The i-th row of a $|G| \times |\mathcal{P}|$ matrix is called a* code *of object g_i.*

Hence, each object is characterized by the corresponding code $c \in \{0,1\}^k$, i.e., a row of the shattering matrix. The set of objects having the same code c we denote by $G(\mathcal{P}, c)$, its cardinality is denoted by $|G(\mathcal{P}, c)|$.

However, instead of computing the VC-dimension, in pattern mining, one rather evaluates the informativeness (diversity) of patterns by computing the joint entropy of this "shattering matrix". The *joint entropy* [KH06a] is given by

$$H(\mathcal{P}) = -\sum_{c \in \{0,1\}^k} \frac{|G(\mathcal{P}, c)|}{|G|} \log \frac{|G(\mathcal{P}, c)|}{|G|}.$$

as it was mentioned in the previous section, we let $0 \log 0 = 0$.

The joint entropy does not take into account the cases where some patterns are reducible, i.e., when removal of a pattern does not change the splitting of objects. To make this measure sensitive to repetitive patterns, one may normalize the joint entropy by the size of the pattern set. The *normalized joint entropy* is given by

$$H_N(\mathcal{P}) = \frac{H(\mathcal{P})}{|\mathcal{P}|}.$$

Example. Let us consider how the joint entropy is computed using a small dataset from Fig. 2.5 (left). The signatures of patterns from the set $\mathcal{P} = \{abc, ab, abfg, defg, fg\}$ are columns of the shattering

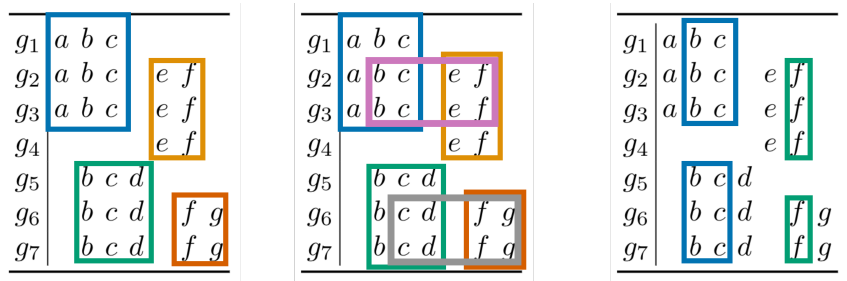


Figure 2.6: Covering by pattern sets $\mathcal{P}_1 = \{abc, bcd, ef, fg\}$ (left), $\mathcal{P}_2 = \{abc, bcd, ef, fg, bcef, cdfg\}$ (middle), and $\mathcal{P}_3 = \{bc, f\}$ (right). The exclusive coverage of all pattern sets is equal to $\{g_1, g_4, g_5\} = 3$

matrix given in Fig. 2.5 (right). The set of objects is split into 4 groups w.r.t. the codes, namely $G(\mathcal{P}, 11000) = \{g_1, g_2\}$, $G(\mathcal{P}, 01000) = \{g_3, g_4\}$, $G(\mathcal{P}, 01101) = \{g_5\}$, and $G(\mathcal{P}, 00011) = \{g_6, g_7\}$.

Pattern abc shatters objects into 2 subsets $\{g_1, g_2\}$ and $\{g_3, \ldots, g_7\}$. The pattern set $\{abc, ab\}$ shatters the objects into 3 subsets: $\{g_1, g_2\}$, $\{g_3, g_4, g_5\}$, and $\{g_6, g_7\}$, since they are characterized by the following codes (rows): 11, 01, and 00 in the corresponding $|G| \times |\{abc, ab\}|$-shattering matrix. Thus, its joint entropy is $H(\{abc, ab\}) = -\frac{2}{7}\log\frac{2}{7} - \frac{3}{7}\log\frac{3}{7} - \frac{2}{7}\log\frac{2}{7} = 1.56$, and the normalized entropy is equal to $H_N(\{abc, ab\}) = 1.56/2 = 0.78$. The pattern set $\{abc, ab, defg\}$ shatters objects in the same way as the previous pattern set, thus $H(\{abc, ab, defg\}) = 1.56$, however, the normalized joint entropy is lower and equal to $H_N(\{abc, ab, defg\}) = 1.56/3 = 0.52$. The joint entropy of the whole set $\mathcal{P} = \{abc, ab, defg, defg, fg\}$ is given by $H(\mathcal{P}) = 3 \cdot (-\frac{2}{7}\log\frac{2}{7}) - \frac{1}{7}\log\frac{1}{7} = 1.95$, the normalized joint entropy is equal to $H_N(\mathcal{P}) = 1.95/5 = 0.39$

Exclusive coverage. *Exclusive coverage* [KH06b] was proposed to assess overlaps among the patterns. The exclusive coverage takes into account only the objects that are covered by a single pattern. Let $G(p) \subseteq G$ be a set of objects covered by a pattern p, then the set of objects that are covered by a set of patterns $\mathcal{P}$ is given by $G(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} G(p)$. The exclusive coverage is computed as follows:

$$EC = \sum_{p \in \mathcal{P}} (|G(p)| - |G(p) \cap G(\mathcal{P} \setminus \{p\})|) \in [0, |G|].$$

The main drawback of this measure is that it counts the object covered exclusively by patterns, however, if the same object is covered by multiple patterns, the exclusive coverage does not indicate how redundant the coverage is.

Example. Let us consider the limitations of the exclusive coverage using a small example from Fig. 2.6. The exclusive coverage of all pattern sets is the same, i.e., $EC(\mathcal{P}_1) = EC(\mathcal{P}_2) = EC(\mathcal{P}_3) = |\{g_1, g_4, g_5\}| = 3$. However, $\mathcal{P}_1$ (on the left) is less redundant than $\mathcal{P}_2$, while $\mathcal{P}_3$ does not cover the whole dataset, and, intuitively, does not describe well the dataset.

Overlap rate. To tackle the aforementioned issue, we propose another measure that takes into account overlaps in data fragments rather than in object sets. Let $\mathcal{P}$ be a set of itemsets. Then the overlapping rate is given by

$$OV(\mathcal{P}, D) = \frac{cover(\mathcal{P}, D)}{\sum_{X \in \mathcal{P}} area(X)} \in [1, |\mathcal{P}|), \tag{2.2}$$

where $cover(\mathcal{P}, D)$ is the area in dataset D covered by patterns from $\mathcal{P}$, $area(X)$ is the pattern area, i.e. $|X| \cdot |X'|$. In other words, OV is the average number of patterns that cover the covered items (cells) in the dataset. For a non-redundant pattern set the overlapping rate is equal to 1, i.e., each item is covered by a single pattern.

Example. Let us consider the overlapping rate of pattern sets using the running example from Fig. 2.6. Informally speaking, to compute OV one needs to count the average number of patterns covering the covered cells in the dataset. The overlapping rates of the rightmost and leftmost pattern sets are the

same and equal to $OV(\mathcal{P}_1) = OV(\mathcal{P}_3) = 1$ (since $28/28 = 17/17$), $OV(\mathcal{P}_2) = 48/28 = 1.71$. Thus, OV captures the redundancy in covering unlike EC, however, does not characterize how many items are covered by patterns. For example, the quality of $\mathcal{P}_1$ and $\mathcal{P}_3$ are indistinguishable, while, intuitively, $\mathcal{P}_1$ is better than $\mathcal{P}_3$. To estimate how well a pattern set covers the data, we introduce the next measure.

Uncovered (cell) ratio. The uncovered cell ratio is the relative number of cells (or objects) covered by a pattern set:

$$UR(\mathcal{P}, D) = 1 - \frac{cover(\mathcal{P}, D)}{||D||} \in [0, 1], \tag{2.3}$$

where $||D||$ is the number of non-empty cells, i.e., of item occurrences, in the dataset D. The lower the value, the better.

Example. Let us consider the ratio of uncovered cells for the pattern sets from the running example from Fig. 2.6. Pattern sets $\mathcal{P}_1$ and $\mathcal{P}_2$ cover entirely the dataset, thus, $UR(\mathcal{P}_1, D) = UR(\mathcal{P}_2, D) = 0$. The ratio of uncovered cells is higher for the pattern set $\mathcal{P}_3$ and is equal to $UR(\mathcal{P}_3, D) = 1 - 17/28 = 9/28$.

The considered examples show that, to evaluate versatile qualities of pattern sets, it is preferable to use multiple measures since one measure does not allow for assessing several characteristics, e.g., redundancy, descriptiveness, diversity of patterns in the set.

2.4.2 Quality measures in supervised settings

The quality measures considered in the previous section evaluate patterns w.r.t. the dataset and other patterns in the pattern set. To assess how meaningful patterns are, some background knowledge is required. Among possible background knowledge, the class labels are widely available information to perform this task. To evaluate patterns using the class labels, one may adopt quality measures for classifiers or even build a classifier and assess its quality. Here we distinguish the following categories of quality measures applicable in supervised settings:

- measures for a set of *patterns* (i.e., applied to each pattern individually) and a *pattern set* (i.e., applied to the whole set);
- measures to assess how well patterns *describe* the data (i.e., assessment of how patterns "fit" the data) and how well patterns *predict* the classes of new instances (i.e., ability of patterns to generalize the knowledge extracted from the training set);
- measures to evaluate the quality of *patterns themselves* or *classifiers* built based on these patterns.

Regarding the evaluation of the descriptive and predictive ability of patterns, we emphasize that the class labels are not available during pattern mining and are used only for assessing the pattern quality. Thus, the evaluation of patterns with the class labels on the same dataset that they were built (i.e., the "training set") allows us to estimate how well a particular pattern fits the data w.r.t. the class label, while the comparison of the quality of pattern on the training and test set allows for assessing the generalization ability of patterns.

We also emphasize the difference between approaches to evaluation of patterns themselves and classifies built on these patterns. In the first case, we minimize the impact of a classification model and assess patterns "as they are". In the second case, we assess how useful patterns may be for building a classification model. However, there are several details that should be taken into account: (i) a particular classification model may be more suitable for patterns computed by a particular pattern mining method, or even for particular dataset, (ii) the quality of the classifiers may be improved by tuning parameters of classifiers, or by slightly changing the classification model. Hence, in the second case, we do not evaluate patterns directly but rather evaluate them w.r.t. the chosen classification model.

Let us consider the most common approaches. We begin with the simplest one — quality measures applied to patterns themselves.

g_1	a	b	c						+
g_2	a	b	c						+
g_3	a	b							+
g_4	a	b							-
g_5	a	b					f	g	+
g_6					d	e	f	g	-
g_7					d	e	f	g	-

	class label	tp	fp	tn	fn	$F1$	tpr	fpr
abc	+	2	0	3	2	2/3	1/2	0
ab	+	4	1	2	0	8/9	1	1/3
$abfg$	+	1	0	3	3	2/5	1/4	0
$defg$	-	2	0	4	1	4/5	2/3	0
fg	-	2	1	3	1	2/3	2/3	1/4

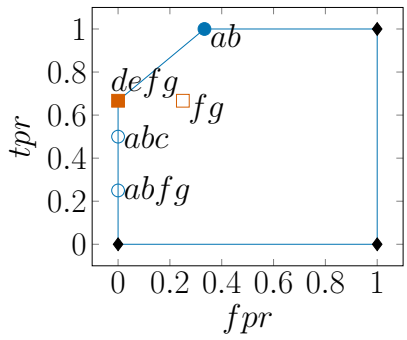


Figure 2.7: Dataset with the class labels (left), supervised characteristics of itemsets w.r.t. the dataset (middle), and the itemset located in the ROC space. The pattern class labels are computed by the majority vote. The convex hull is built on 5 extreme points filled with blue, red, or black colors. Blue round and red square points correspond to patterns labeled as positive and negative, respectively. 3 black diamonds corresponds to the fixed points needed to build the convex hull

Evaluation of patterns by quality measures

In approaches of this group each pattern is evaluated as a classifier. To assign a class label to a pattern we use the majority vote scheme.

Let p be a pattern that describes a set of objects $G(p) \subseteq G$. Each object $g \in G$ belongs to a single class $class(g) \in \mathcal{Y}$. Then the class label of a pattern p is defined as follows:

$$class(p) = \arg\max_{y \in \mathcal{Y}} |\{g \mid g \in G(p), class(g) = y\}|. \tag{2.4}$$

The number of true/false positive/negative instances is defined as follows:

		Actual values	
		positive	negative
Predicted values	positive	$tp(p,G) = \lvert\{g \mid g \in G(p), class(p) = class(g)\}\rvert$	$fp(p,G) = \lvert G(p)\rvert - tp$
	negative	$fn(p,G) = \lvert\{g \mid g \in G \setminus G(p), class(p) = class(g)\}\rvert$	$tn(p,G) = \lvert G \setminus G(p)\rvert - fn$

Then the quality of a pattern p can be evaluated by one of the following measures:

$$F1(p) = \frac{2tp}{2tp + fp + fn},\ accuracy(p) = \frac{tp + tn}{tp + tn + fp + fn},\ precision(p) = \frac{tp}{tp + fp}.$$

These measures characterize purity of the group of objects described by pattern p. Each pattern is evaluated individually. The quality of the whole pattern set can be evaluated by computing the average value, or by applying any other aggregation function. Using the same dataset both for computing patterns and evaluation of their quality (i.e., using one training set), we assess the descriptive ability of patterns. However, the same quality measures may be used to evaluate the predictive ability of patterns. In this case, an additional dataset (i.e., test set) is used. The predictive ability of patterns is evaluated by comparing the values of a chosen quality measure computed on the training and test sets. In particular, one may be interested in patterns where the difference in the quality measures computed based on both sets does not exceed a certain threshold.

Example. Let us consider the running example. Now, each object has also a class label. The dataset and patterns are given in Fig. 2.7. The class labels of patterns $\{abc, ab, abfg, defg, fg\}$ are computed by the majority vote scheme and given on the right. The average value of $F1$ measure is 0.68. This value characterizes the descriptive ability of patterns. Precision, accuracy and other measures are computed in the same way.

To evaluate the predictive ability of the patterns, we consider in addition the set of yet unseen instances, i.e., the test set, from Fig. 2.8 (left). The characteristics of patterns on this set are given in

g_8	a	b			f	g		+
g_9		b	c	e				+
g_{10}	a			e	f	g		-
g_{11}				e	f	g		-

	class label	tp	fp	tn	fn	$F1$	tpr	fpr
abc	+	0	0	2	2	0	0	0
ab	+	1	0	2	1	2/3	1/2	0
$abfg$	+	1	0	2	1	2/3	1/2	0
$defg$	-	0	0	2	2	0	0	0
fg	-	2	1	1	0	4/5	1	1/2

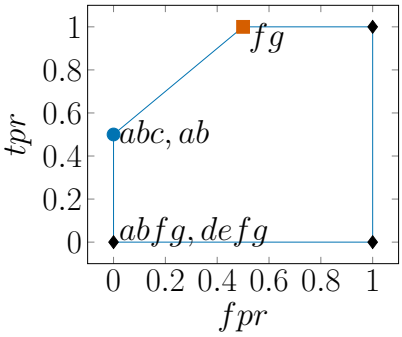


Figure 2.8: Test dataset with class labels (left), supervised characteristics of itemsets on this set, the itemset class labels are computed on the training set (middle), and the itemset locations in the ROC space (right). The convex hull is built on 5 extreme points filled with blue, red, or black colors. Blue round and red square points correspond to itemsets labeled as positive and negative, respectively. The black diamonds are fixed points needed to build the convex hull

the middle. The average F1 measure on the test set is 0.43, which is lower than on the training set, thus, the predictive ability is worse that the descriptive one. In the same way, we may use other measures, as well as to set the "tolerance" threshold on the difference in the performance on the training and test set.

The considered measures evaluate each pattern individually. Let us consider an alternative approach, where the pattern set is considered as a whole and only best patterns are taking into account.

Area under ROC-curve (AUC). To measure AUC one needs to compute for each pattern true positive rate (tpr) and false positive rate (fpr) as follows:

$$tpr = \frac{tp}{tp + fn}, \; fpr = \frac{fp}{fp + tn},$$

and then to plot the corresponding points in the ROC (tpr-fpr) space. The area of the convex hull built on these points and 3 additional points (0, 0), (1, 0), (1, 1) will give the quality of the pattern set. In this approach only the best patterns are involved in the evaluation.

Example. Let us consider how AUC is computed using the running example from Fig. 2.7. To compute it, one needs to draw points corresponding to evaluated patterns in the ROC space. The values of tpr and fpr are given in Fig. 2.7 (middle). The patterns in the ROC space are given in Fig. 2.7 (right), AUC is equal to 8/9. In the same way, one may compute AUC on the test set, see Fig. 2.8 (right), where AUC is equal to 7/8.

Evaluation of patterns by building classifiers

In the literature, there exists a lot of approaches for building classifiers based on patterns. This subject was exhaustively studied in the case where patterns are itemsets. Further, we briefly describe the most common approaches. In this study, we leave aside the approaches for computing patterns w.r.t. an attribute of interest (i.e., class labels), which are usually referred to, in the scientific literature, as *subgroup discovery*.

In [ZN14] the authors distinguish two categories of itemset-based classification approaches, namely *direct* and *indirect*. Let us consider them in detail.

Direct classification. The computed patterns may be used directly as classifiers. A pattern X associated with a class y can be considered as a rule $X \to y$. Then, the obtained rules can be combined into either a *decision list* or an *ensemble*.

When building a decision list, one may order the rules, for example, by confidence (descending), support (descending) and length (ascending), as it is done in CBA [LHM98], or by information gain (descending), as it is done in LAC [VMZ06].

In the ensembles, a pattern "*votes*" for objects. For example, when patterns are itemsets, an itemset votes for an object if it is included in the object description. However, one may use a relaxation, e.g., define the similarity between an itemset and object description, for example, cosine or Jaccard similarity, and consider that an itemset describes an object when the similarity between them is greater than a given threshold. The votes can be weighted by the "quality" of patterns, e.g., FitCare [CGSF+13] and ARC-BC [AZ02] use relative supports per class. A detailed overview of the itemset-based classifiers can be found in [ZN14].

Decision table majority classifier. In [KH06b] the authors adopted the *decision table majority* (DTM) classifier [Koh95] to evaluate pattern sets. The original model of the DTM classifier has two components: a scheme (i.e., a set of selected features) and a body (i.e., labeled objects projected onto the feature space defined by the scheme). Instead of selected features the author of [KH06b] uses the pattern signatures (see Definition 8) and object codes (see Definition 9). To evaluate pattern set $\mathcal{P}$ one needs to compute signatures and for each object code $c \in \{0,1\}^{|\mathcal{P}|}$ estimate its quality as follows:

$$f(\mathcal{P}, c) = \frac{\arg\max_{y \in \mathcal{Y}} |\{g \mid g \in G(\mathcal{P}, c), class(g) = y\}|}{|G(\mathcal{P}, c)|}, \qquad \text{if } |G(\mathcal{P}, c)| > 0$$

$$f(\mathcal{P}, c) = \frac{\arg\max_{y \in \mathcal{Y}} |\{g \mid g \in G, class(g) = y\}|}{|G|}, \qquad \text{if } |G(\mathcal{P}, c)| = 0.$$

Then, the classification of unlabeled objects is performed by finding the exact matching with the code in the decision table. DTM can be used to assess the pattern set quality by computing the accuracy of the DTM classifier by means of cross-validation, as it was proposed in [KH06b].

However DTM has some limitations, namely it works better when patterns have a low cardinality, and only a small number of the "best patterns" is used [KH06a]. In this work we consider ensembles of classifiers as more flexible approach to evaluation of the predictive ability of patterns.

Ensembles of classifiers. Let $\mathcal{P}$ be a set of patterns, where each pattern $p \in \mathcal{P}$ is an itemset with an associated class label. The class labels can be defined, for example, by the majority vote scheme (Equation 2.4). The weights of patterns may be set based on their (un)supervlised characteristics, e.g., support $w(p) = (tp + fp)/|G|$, precision $w(p) = tp/(tp + fp)$, etc. The predicted class of an object is defined by the weighted majority vote scheme, i.e.,

$$class(g, \mathcal{P}) = \arg\max_{y \in \mathcal{Y}} \sum_{\substack{p \in \mathcal{P} \\ class(p) = y}} \{w(p) \mid p \subseteq \{g\}', g \in G\}.$$

$$class(g, \mathcal{P}) = \begin{cases} \arg\max_{y \in \mathcal{Y}} \sum_{\substack{p \in \mathcal{P} \\ class(p) = y}} \{w(p) \mid p \subseteq \{g\}', g \in G\}, & \text{if } \exists p \in \mathcal{P} \text{ such that } |G(p)| > 0 \\ \arg\max_{y \in \mathcal{Y}} |\{g \mid g \in G(p), class(g) = y\}|, & \text{otherwise.} \end{cases}$$

In the case where no pattern has voted, the class label of the majority class in the training set is assigned to g.

Example. Let us consider the set of itemsets $\{abc, ab, abfg, defg, fg\}$ from the running example from Fig. 2.7. The test set and the pattern performance on it are given in Fig. 2.8. The default class is "+" since the majority of objects in the training set belongs to the "+" class. Thus, in the case where there are no voted patterns, the object is classified as a "+"-instance. Let ensemble of classifiers be composed of itemsets ab and fg weighted by precision: $w(ab) = 4/5$, $w(fg) = 2/3$. Then, $class(g_8, \mathcal{P})$ ="+", since $4/5 > 2/3$; $class(g_9, \mathcal{P})$ ="+", as the default class; $class(g_{10}, \mathcal{P}) = class(g_{11}, \mathcal{P})$ ="−", since the only pattern voted for them is "fg".

The problem of building high-quality pattern-based classifiers remains out of the scope of this work, since it involves additional study of the version space and strategies for choosing the best classifiers. The interested readers are referred to the studies of the version space [Mit97] and the JSM-method [Kuz91].

Indirect classification. The indirect classification models imply patterns be used within more complex models. For example, compression-based pattern miners (e.g., KRIMP [VVLS11] and SLIM [SV12], that we consider in detail in Section 4.2) return a code table, which contains selected patterns and associated lengths of code words. To build a classifier the dataset is split into several subdataset w.r.t. the class labels. The code tables are computed for each subdataset independently. To classify an unlabeled object, it is compressed using each the code tables. Then, the class of the code table that ensure the shortest encoding is assigned to the object.

Another "indirect" usage of itemsets as classifiers was proposed in [MW99] and is called LARGE BAYES. The authors generalize the Naïve Bayes approach and use itemsets instead of single items to compute the probability of objects in a given class. LARGE BAYES takes as interesting itemsets those frequent itemsets whose probability cannot be accurately approximated by their direct subsets. To estimate the object probability LARGE BAYES use the largest (by inclusion) interesting patterns. The authors leverage the *product approximation* [LI59], i.e., the itemsets are combined using the chain rule of probability.

However, being built on the same pattern set, some classifiers may provide better quality than the others. In this work, we do not focus on building the classifiers that outperform the state-of-the-art classifiers, but rather use them to assess the quality of pattern sets.

Thus, in this section we consider some approaches to evaluate patterns in supervised settings. We discussed two options, namely assessing *descriptive* and *predictive* quality of patterns. Technically, the key difference between the aforementioned options is that in the first case we evaluate patterns on the set of objects that they were computed (however, the object labels are unavailable during pattern mining) and in the second case on the test set, i.e., new unseen yet objects. In Chapters 4 and 5 we use the described techniques to evaluate the quality of the developed pattern miners.

Note also that in this section, discussing the classifiers, we used closed itemsets, i.e., the largest itemsets in the equivalence class. Intuitively, the "best classifiers" are in the middle of the equivalence class. For example, in the running example, a set of three itemsets, namely $ab, bc \in Equiv(abc)$ and $efg \in Equiv(defg)$ ensure the best accuracy on the test set, while the set of the corresponding closed itemsets abc, $defg$ does not. We emphasize that the choice of the itemsets in the equivalence class that ensure the best predictive ability is not a matter of this study. Here we focus on pattern mining and use the largest (closed) patterns in the equivalence classes.

3

Closure structure

Contents

3.1 Introduction **24**
3.2 Related work **25**
3.3 Basic notions **26**
3.4 Closure structure and its properties **28**
3.4.1 Keys and passkeys 28
3.4.2 Closure structure 29
3.4.3 Passkey-based order ideal 30
3.4.4 Assessing data complexity 30
3.4.5 Closeness under sampling 31
3.5 The GDPM algorithm **32**
3.5.1 Computing the closure structure with GDPM 32
3.5.2 The extent-based version of GDPM 34
3.5.3 Complexity of GDPM 35
3.5.4 Related approaches to key-based enumeration 35
3.5.5 Computing passkeys. Towards polynomial complexity 37
3.6 Experiments **39**
3.6.1 Characteristics of the datasets 39
3.6.2 Computational performance 40
3.6.3 Data topology or frequency distribution within levels 42
3.6.4 Coverage and overlaps 45
3.6.5 Usefulness of concepts 45
3.6.6 Case study 46
3.7 Discussion and conclusion **49**

In this chapter, we are revisiting pattern mining and especially itemset mining, which allows one to analyze binary datasets, in search for interesting and meaningful association rules and itemsets in an unsupervised way. We introduce a concise representation — the closure structure — based on closed itemsets and their minimum generators, for capturing the intrinsic content of a dataset. The closure structure allows one to understand the topology of the dataset as the whole and the inherent complexity of the data. We propose a formalization of the closure structure in terms of formal concept analysis, which is well adapted to study this data topology. We present and demonstrate theoretical results, as well as an algorithm that we refer to as GDPM (Gradual Discovery in Pattern Mining). GDPM is rather unique in its functionality as it returns a characterization of the topology of a dataset in terms of complexity levels, highlighting the diversity and the distribution of the itemsets. Finally, a series of experiments shows how GDPM can be practically used and what can be expected from the output.

3.1 Introduction

In this chapter, we are revisiting pattern mining and especially itemset mining. Itemset mining allows to analyze binary datasets in search for interesting and meaningful association rules and respective itemsets in an unsupervised way [HPK11]. There are two main existing approaches to itemset mining (IM) which are based on (i) an exhaustive approach and (ii) sampling approach.

The majority of the exhaustive approaches are based on enumeration of itemsets w.r.t. frequency. Over the last decades, a large variety of algorithms for mining frequent itemsets was proposed, e.g., join-based algorithms [AS94] which rely on a level-wise exploration, where $(k+1)$-itemsets are derived from k-itemsets, vertical exploration [SHS+00], projection-based techniques [HPY00], etc. This intensive development of algorithms of frequent itemset mining (FIM) is mainly due to the fact that frequent itemsets have an intuitive interpretation, namely "the most interesting itemsets are frequently occurring in the data". Moreover, frequency is an antimonotonic measure w.r.t. the size of the itemset and this allows to enumerate very efficiently all frequent itemsets and only them. However, frequent itemset mining shows several drawbacks as frequent itemsets are rather not (necessarily) interesting and in general very redundant. In addition, when the frequency threshold decreases, the problems of combinatorial explosion and itemset redundancy become more acute.

Focusing on closed itemsets [PBTL99b, ZO98] allows for a significant reduction of the number of itemsets by replacing a whole class of itemsets having the same support with the largest one, i.e., the corresponding closed itemset. Nowadays, there exist very efficient algorithms for computing frequent closed itemsets [HI17, UKA05]. Even for a low frequency threshold, they are able to efficiently generate an exponential number of closed itemsets. However, the efficient generation of closed itemsets solves the problem of the exponential explosion of itemsets only partially since the main difficulties appear after, when the generated itemsets are processed.

In the second approach, i.e., an alternative to the exhaustive enumeration of itemsets, more recent algorithms are based on sampling [DvLDR17] and on a gradual search for itemsets according to an interestingness measure or a set of constraints [SV12]. For example, Sofia is an algorithm which starts from one itemset using derivation rules and outputting a set of itemsets organized within a lattice and satisfying the constraints related to the derivation rules [BKN15a]. These algorithms usually output a rather small set of itemsets while they may provide only an approximate solution.

Considering both approaches, we can remark that, even if they use quite different techniques, i.e., exhaustive enumeration on the one hand and sampling on the other hand, these approaches rely on the same assumption, namely that the intrinsic structure of the dataset under study can be understood by means of selecting itemsets w.r.t. a subjective interestingness measure or a set of constraints. Then, in each approach, a particular set of itemsets is returned, which offers a multifaceted view about the data, but does not provide an "intrinsic structure" underlying the data. This is precisely such a structure that we introduce and study hereafter.

In this chapter we propose a method and an algorithm for revealing and understanding the intrinsic structure of a dataset, called the *closure structure*, which shows the distribution of the itemsets in the data in terms of frequency, and in addition supports an interpretation of the content of a dataset. This closure structure is based on the closed itemsets and minimum elements of their equivalence classes in an unsupervised setting, and may be computed independently of any interestingness measure or set of constraints. Actually, the closed itemsets are the base of what we call the "topology of the dataset", replacing the traditional open sets in mathematical topology. Closed itemsets play the same role as open sets and are used to cover the dataset in a specific way, giving rise to the closure structure, which is based on a partition of levels, induced by the set of equivalence classes related to these closed itemsets.

The closure structure and its levels provide a representation of the complexity of the content of a dataset. We propose a formalization of the closure structure in terms of formal concept analysis [GW99], which is well adapted to the study of closed itemsets, equivalence classes, and data topology. We present and demonstrate theoretical results about the closure structures, among which a characterization of the closure structure and complexity results about itemset mining, and as well as a lower bound on the size of the pattern space — based on a lattice — which is a good indicator of the complexity of the dataset.

We introduce the GDPM algorithm for "Gradual Discovery in Pattern Mining" which computes the closure structure of a dataset. Given a dataset, the GDPM algorithm returns a characterization of the

levels constituting the closure structure of this dataset. We study the complexity of the algorithm and compare it with related algorithms. We also show in a series of experiments how GDPM can be used and what can be expected. For example, it is possible to determine which are the levels where the itemsets are most diverse, how the frequency is distributed among the levels, and when a search for interesting itemsets can be stopped without loss. Then we propose a comparative study on the intrinsic complexity and the topology using a collection of public domain datasets. To the best of our knowledge, GDPM is rather unique in its functionality as it returns a characterization of the topology of a dataset in terms of the closure structure, allowing to guide the mining of the dataset.

3.2 Related work

In data mining and knowledge discovery computing itemsets is of main importance since itemsets are used to compute (i) a succinct representation of a dataset by a small set of interesting and non-redundant patterns, (ii) a small set of non-redundant high-quality association rules.

There are several approaches to compute itemsets: (i) an exhaustive enumeration of all itemsets followed by a selection of those satisfying imposed constraints [VT14], (ii) a gradual enumeration of some itemsets guided by an objective (or by constraints) [SV12], (iii) mining top-k itemsets w.r.t. constraints [MVT12], (iv) sampling a subset of itemsets w.r.t. a probability distribution that conforms to an interestingness measure [BLPG11, BMG12]. To reduce redundancy when enumerating itemsets, the search space can be shrunk to *closed itemsets*, i.e., the maximal itemsets among those that are associated with a given set of objects (support). Meanwhile, in sampling techniques with rejection, this restriction of arbitrary itemsets to closed ones may increase the sampling time [BGG10].

Generally, two types of data can be defined: (i) "deterministic", where the object-attribute relations reflect non-probabilistic properties and where the noise rate is low, and (ii) "probabilisic", where the relations between objects and attributes are of stochastic nature. For example, deterministic data include the profiles of users with features such as education, gender, marital status, number of children, etc. By contrast, probabilistic data may describe basic item lists, where each item is present or absent with a certain probability (since one could forget about an item or to buy an item spontaneously).

For probabilistic data, enumerating closed itemsets may be infeasible in practice, since the number of closed itemsets may grow exponentially fast w.r.t. the noise rate [KOR10]. However, for "deterministic" data it might be very important to have a complete set of itemsets that meet user-defined constraints or to get a set of implications (i.e., association rules with confidence 1). In the last decades, substantial efforts in different directions have been made to establish a theoretical foundation of closed itemset mining within several theoretical frameworks: formal concept analysis [GW99], set systems [BHPW07a, BHPW07b, BHPW10] and data mining [PBTL99b, PBTL99c, PTB+05].

A main challenge in closed itemset mining is computational complexity. The number of closed itemsets can be exponential in the dataset size. In [Kuz93] an efficient depth-first search algorithm, called CbO, for computing closed itemsets with polynomial delay was proposed. To avoid enumerating the same itemsets several times CbO relies on a canonicity test. Loosely speaking, a closed itemset I_1 passes the canonicity set if an itemset I_2 that generates I_1 is lexicographically smaller (we introduce this notion in Section 3.5.1) than I_1 itself. The execution trace of CbO is a spanning tree of the lattice composed of all closed itemsets ordered by set-inclusion. Further, other successor algorithms were developed within the FCA framework [KO02]. One of the most efficient algorithm from the CbO family is In-Close [And14], where the canonicity is leveraged to compute a partial closure, i.e., checking the closure on a subset of attributes that potentially may not pass the canonicity test. The latter allows for reducing the running time.

Then Uno and Arimura proposed the LCM algorithm [UAUA04], which computes all frequent closed itemsets with polynomial-delay and polynomial-space complexity and is quite efficient in practice [UKA04]. However, LCM requires a frequency threshold. The successor of LCM, the Alpine algorithm [HI17], does not have this drawback. It computes (closed) itemsets of decreasing frequency gradually and ensures completeness of the results, i.e., being interrupted anytime it returns a complete set of frequent (closed) itemset for a certain frequency threshold. Recently, in [JKK20] it was shown that LCM is basically CbO with some implementation tricks ensuring to LCM a very good performance, e.g., "occurrence

deliver" (storing the attribute extensions in arraylists to avoid multiple traversal of a dataset), usage of "conditional databases" (reducing the dataset size in recursive calls by removing "irrelevant" objects and attributes and merging repetitive objects).

In [KV09] the authors study how different data structures, e.g., bitarrays, sorted linked lists, hash tables and others, affect the execution time. In the experiments, it was shown that binary search trees and linked lists are appropriate data structures for large sparse datasets, while bitarrays provide better execution time for small or dense datasets.

The theoretical aspects of closed itemset computing were studied in parallel within the framework of *set systems*, where some elements commonly used in FCA were re-discovered. For example, in [BHPW07a] the authors consider inductive generators and prove that every closed itemset (except $\emptyset$) has an inductive generator. Roughly speaking, it means that for each closed itemset B there exists a closed itemset B' and an item $e \in B \setminus B'$ such that the closure of $B' \cup \{e\}$ is equal to B. It follows directly from this definition that inductive generators are parents of the closed itemsets in the spanning tree used by the CbO algorithm. In [AU09] a polynomial-delay and polynomial-space algorithm, called CloGenDFS, was proposed. As CbO, this algorithm avoids the explicit duplication test for itemsets. The execution tree used by CloGenDFS (called a family tree) is quite similar to that used by CbO in the bottom-up strategy.

The results described above are related to *computational aspects* of closed itemset mining. However, they do not address the problem of discovering an *intrinsic structure* underlying the set of closed itemsets and, more generally, the dataset itself. In a seminal work related to this direction [PBTL99b, STB+02], it was proposed to consider minimal (by set-inclusion) itemsets among all itemsets that are associated with the same set of objects, i.e., minimal generators. These sets are used to find a non-redundant set of association rules and implications (rules with confidence 1) that summarizes the data. In this work we consider minimum (by size) itemsets and use them to characterize this intrinsic structure and thus the data complexity.

The minimum implication base [GW99] — also known as Duquenne-Guigues basis [GD86] — is one first way of characterizing the "complexity" of the set of closed itemsets w.r.t. implications. This basis represents the minimal set of implications sufficient to infer all other implications using the Armstrong rules [Arm74]. Computing a minimal basis of implications is of particular importance in database theory, because there exists a one-to-one correspondence between the functional dependencies in an original database and implications computed on an associated database [GW99, CLN14, BCKN18].

Besides computing the minimal set of implications, there exists a range of approaches to computing non-redundant (not necessarily minimal in the sense of inference) set of implications or association rules. For example, in [Zak00a] the authors propose an approach to compute a set of *general* implications and association rules based on closed itemsets. General rules have the smallest antecedent and consequent among the equivalent ones. In [BTP+00] an alternative approach for computing non-redundant implications and association rules was proposed. Each rule in this approach has the smallest antecedent and the largest consequent among the equivalent ones. And these rules are considered as "good rules", especially for interpretation purposes.

All approaches mentioned above rely implicitly or explicitly on an intrinsic structure, however this structure was rarely if ever formalized, and its properties have not yet been studied in depth.

In this chapter, we define a level-wise structure for representing the "intrinsic structure" of a dataset w.r.t. closed itemsets, that we call "*closure structure*". We define the "complexity" of closed itemsets w.r.t. the size of minimal/minimum itemset common to a set of itemsets equivalent to the closed itemset itself. The levels of the closure structure are composed of closed itemsets having the same complexity. We also discuss computational issues, as well as theoretical and empirical properties of the closure structure.

3.3 Basic notions

In this chapter, we deal with binary datasets within the FCA framework. Its basics are given in Section 2.1. Here we briefly recall the key notations and introduce new ones.

A *formal context* is a triple $\mathbb{K} = (G, M, I)$, where G is a set of objects, M is a set of attributes and $I \subseteq G \times M$ is the incidence relation, i.e., $(g, m) \in I$ if object g has attribute m.

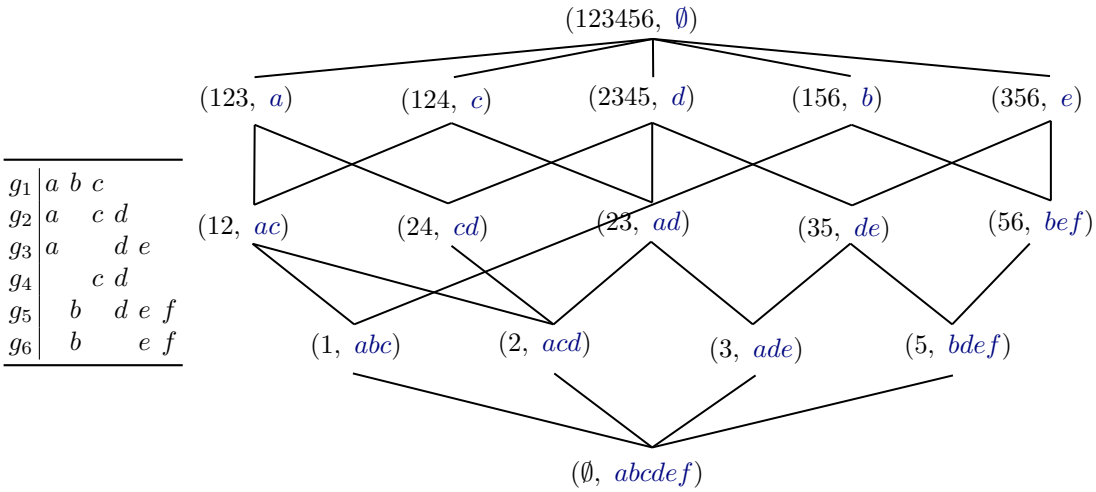


Figure 3.1: A dataset of transactions (left) and its concept lattice (right)

Two derivation operators $(\cdot)'$ are defined for $A \subseteq G$ and $B \subseteq M$ as follows:

$$A' = \{m \in M \mid \forall g \in A : gIm\}\,,\ B' = \{g \in G \mid \forall m \in B : gIm\}\,.$$

For $A \subseteq G$, $B \subseteq M$, a pair (A, B) such that $A' = B$ and $B' = A$, is called a *formal concept*, then A and B are closed sets and called *extent* and *intent* (or *closed itemsets*), respectively. The number of items in an itemset B is called the *size* or *cardinality* of B.

We distinguish maximal/minimal and maximum/minimum itemsets. Let $\mathcal{S}$ be a set of itemsets. We call an itemset $X \in \mathcal{S}$ *maximal* w.r.t. $\mathcal{S}$ if there is no other itemset in $\mathcal{S}$ that includes X. We call an itemset $X \in \mathcal{S}$ *minimal* if there is no other itemset in $\mathcal{S}$ that are included into X. An itemset is called *maximum* in $\mathcal{S}$ if it has the largest *size* or *cardinality*, i.e., consists of the maximal number of items. Minimum itemset is defined dually.

Given an itemset X, its equivalence class $Equiv(X)$ is the set of itemsets with the same extent, i.e., $Equiv(X) = \{Y \mid Y \subseteq X, X' = Y'\}$. A *closed itemset* X'' is the maximum (largest cardinality) itemset in the equivalence class $Equiv(X)$. Thus, the concept lattice concisely represents the whole pattern search space, i.e., all possible combinations of items, where itemsets having the same support are encapsulated in the corresponding closed itemsets [PBTL99b].

A *key* $X \in Equiv(B)$ is a minimal generator in $Equiv(B)$, i.e., for every $Y \subset X$ one has $Y' \neq X' = B'$. We denote the set of keys (*key set*) of B by $Key(B)$. Hence, every proper subset of a key is a key that has a different closure [STB+02]. An equivalence class is *trivial* if it consists of only one closed itemset.

Definition 10. *An itemset $X \in Key(B)$ is called a* passkey *if it has the smallest size among all keys in $Key(B)$. We denote the set of passkeys by $pKey(B) \subseteq Key(B)$. For a closed itemset B the* passkey set *is given by $pKey(B) = \{X \mid X \in Key(B), |X| = \min_{Y \in Key(B)} |Y|\}$.*

In an equivalence class, there can be several passkeys but only one closed itemset. All passkeys have the same size in an equivalence class. The size of the closed itemset is maximum among all itemsets in this equivalence class, by definition.

Example. Let us consider the dataset in Fig. 3.1 (the same dataset was considered in Section 2.1, Fig. 2.1). Among all 16 closed itemsets only five have non-trivial equivalence classes, namely *ade*, *abc*, *bef*, *bdef* and *abcdef*. Their keys and passkeys are listed in Table 3.1. The remaining itemsets have trivial equivalence classes, namely $Equiv(B) = \{B\}$. Moreover, every closed itemset B with the trivial equivalence class has $|B|$ immediate subsets that are closed itemsets. For example, itemset *acd* has the trivial equivalence class, and all its immediate subsets *ac*, *ad* and *cd* are closed. In Section 3.4.3 (Proposition 1) we analyze this property.

Table 3.1: Non-trivial equivalence classes of closed itemsets for the dataset from Fig. 3.1. Closed itemsets are given in column B

B	$pKey(B)$	$Key(B)$	$Equiv(B)$
ade	ae	ae	ae, ade
abc	ab, bc	ab, bc	ab, bc, abc
bef	f	f, be	f, be, ef, bf, bef
$bdef$	bd, df	bd, df	bd, df, def, bdf, bde, $bdef$
$abcdef$	af, ce, cf	af, ce, cf, abd, abe, bcd	af, ce, cf, abd, abe, abf, ace, acf, adf, aef, bcd, bce, bcf, bde, bdf, cde, cdf, cef, def, ..., $abcdef$*

* The equivalence class includes all itemsets that contain a key from $Key(abcdef)$.

Among closed itemsets with non-trivial equivalence classes, only bef and $abcdef$ have passkey sets that differ from the key sets, i.e., $pKey(bef) = \{f\}$, $Key(bef) = \{f, be\}$, $pKey(abcdef) = \{af, ce, cf\}$, $Key(abcdef) = pKey(abcdef) \cup \{abd, abe, bcd\}$. The keys from $Key(B)$ are minimal by set-inclusion, i.e., there is no element in the equivalence class that is contained in a key. For example, for key $be \in Key(bef)$ there is no itemset in $Equiv(bef)$, except be itself, that is included in be. The passkeys are keys that have the smallest size. Closed itemset bef has one passkey f. Below, in Fig. 3.9, we give an example of a case with unique passkey and exponentially many keys.

Thus, a passkey is the shortest representative of the equivalence class, i.e., the smallest lossless description (meaning that the whole lattice can be rebuilt using the set of passkeys). Let us consider how passkeys can be used to define the closure structure.

3.4 Closure structure and its properties

3.4.1 Keys and passkeys

When closed itemsets are ordered by set inclusion, the structure containing the whole set of closed itemsets is a complete lattice called the concept lattice (see example in Figure 3.1). However, in a lattice, one is usually interested in local positions of the closed itemsets, or concepts, and most often in direct neighbors [BKN14b]. Usually, the global position of the concepts w.r.t. the largest and smallest elements is not considered as useful information. Moreover, for a closed itemset, there can be exponentially many — w.r.t. its size and support — different ways to reach it from the top or the bottom of the lattice.

In this section, we introduce the closure structure. Unlike the concept lattice, the closure structure is composed of levels that reflect the "intrinsic" complexity of itemsets. The complexity of a closed itemset X is defined by the size of its passkeys, i.e., cardinality minimum generators of X.

In the previous sections we consider different elements of the equivalence class, namely, generators, minimal generators (keys) and minimum generators (passkeys). We can organize the itemset space into levels based on minimal generators. In Fig. 3.2 we show such a splitting. A closed itemset B is located at level k if there exists a key of B of size k. The keys of closed itemsets that differ from the corresponding closed itemsets are given in parentheses. Some closed itemsets have more than one key, e.g., abc has keys ab and bc, $bdef$ has keys bd and df. The problem is that a closed itemset may be located in several levels. For example, bef appears at the 1st and 2nd levels since it has keys f and be, closed itemset $abcdef$ belongs to the 2nd and 3rd levels.

To eliminate this kind of redundancy, we propose to order the set of closed itemsets w.r.t. passkeys or "cardinality minimum generators". Since all the passkeys of a closed itemset have the same size, the closed itemset belongs only to one level, and hence the corresponding organization is a partition.

Let us consider this structure more formally.

Level 1	a	b	c	d	e	bef (f)				
Level 2	abc (ab, bc)	$bdef$ (bd, df)	$abcdef$ (af, cf, ce)		ade (ae)	bef (be)	ac	de	ad	cd
Level 3	acd	$abcdef$ (abd, abe, bcd)								

Figure 3.2: Splitting a set of closed itemsets from Fig. 3.1 by levels w.r.t. the size of their keys. The keys that differ from their closures are given in parentheses. The set-inclusion relations between itemsets are not shown

3.4.2 Closure structure

As it was mentioned above, all passkeys of a closed itemset have the same size. A passkey is a minimal generator of minimum cardinality. A passkey corresponds to the shortest itemset among all itemsets composing an equivalence class and characterizes the complexity of a closed itemset in terms of its size.

Let $\mathcal{C}$ be the set of all closed itemsets for the context (G, M, I), and $pKey(B)$ be the passkey set of a closed itemset $B \in \mathcal{C}$. The function $Level : \mathcal{C} \to \{0, \ldots, |M|\}$ maps a closed itemset B to the size of its passkeys, i.e., $Level(B) = |X|$, where $X \in pKey(B)$ is an arbitrary passkey from $pKey(B)$. The closed itemsets having passkeys of size k constitute the closure level k.

Definition 11. *Let $\mathcal{C}$ be a set of all closed itemsets for a context (G, M, I). Then the* closure level $\mathcal{C}_k$ *is the subset of all closed itemsets with passkeys of size k, i.e., $\mathcal{C}_k = \{B \mid B \in \mathcal{C}, \exists D \in pKey(B), |D| = k\}$.*

In the definition above, in order to define the closure level of a closed itemset B, we use an arbitrary passkey $D \in pKey(B)$. We distinguish passkeys only when we compute the closure structure. We discuss these details later in Section 3.5.

Proposition 1. *The set of closure levels of the context (G, M, I) defines a partition over the set of closed itemsets $\mathcal{C}$ of (G, M, I) into sets of itemsets of increasing complexity w.r.t. the size of the passkeys.*

Proof. It follows from the definition of passkey that the passkeys associated with a closed itemset are all of the same size. Thus, each closed itemset belongs to only one closure level, and the closure levels make a partition. □

Definition 12. *The partition based on the set of closure levels of the context (G, M, I) is called the* closure structure *of (G, M, I).*

The last non-empty level of the closure structure contains the largest passkeys. The size of such keys is the upper bound on the size of the generators of all closed itemsets in a given dataset, i.e., the least upper bound on the sizes of generators of the most "complex" closed itemsets.

Definition 13. *The* closure index *of $\mathcal{C}$, denoted by CI, is the maximal number of non-empty levels of the closure structure, i.e., $CI = \max\{k \mid k = 0, \ldots, |M|, \mathcal{C}_k \neq \emptyset\}$.*

The closure index CI is the largest size of an itemset "sufficient" to represent any closed itemset. More formally, $\forall B \in \mathcal{C}$, $\exists X \in \mathcal{P}(M), |X| \leq CI$ such that $X'' = B$, where $\mathcal{P}(M)$ is the powerset of M. In such a way, the closure index characterizes the complexity of the whole set of closed itemsets $\mathcal{C}$.

Example. Let us consider the closure structure given in Fig. 3.3. Here, in contrast to the situation in Fig. 3.2, each closed itemset appears only once. A closed itemset from the k-th level may still have several passkeys of size k, but all these keys are of the same size. For example, abc has two passkeys: ab and bc, closed itemset $abcdef$ has three passkeys, i.e., af, cf and ce.

$\mathcal{C}_1$	a	b	c	d	e	bef (f)			
$\mathcal{C}_2$	abc (ab, bc)	$bdef$ (bd, df)	$abcdef$ (af, cf, ce)		ade (ae)	ac	de	ad	cd
$\mathcal{C}_3$	acd								

Figure 3.3: The closure structure for the dataset from Fig. 3.1. The passkeys that differ from their closures are highlighted in gray. The set-inclusion relations between itemsets are not shown

3.4.3 Passkey-based order ideal

In [STB+02] it was shown that keys are downward closed, i.e., every proper subset of a key is a key. The latter means that given a certain key X we may reconstruct $2^{|X|} - 1$ other keys without computing any concept. In order theory the structure possessing this property is called an order ideal.

Definition 14. *Let $(\mathcal{P}, \leq)$ be a poset. A subset $\mathcal{I} \subseteq \mathcal{P}$ is called an* order ideal *if $x \in \mathcal{I}$, $y \leq x$ then $y \in \mathcal{I}$.*

So, the set of all keys makes an order ideal in the set of all itemsets ordered by inclusion. The same holds for the set of all passkeys.

Proposition 2. *Let $\mathcal{I}_P = \bigcup_{k=1,\ldots,|M|}\{D \in pKey(B) \mid B \in \mathcal{C}, |D| = k\}$ be the set of passkeys for the context (G, M, I). Then $\mathcal{I}_P$ is an order ideal w.r.t. $\subseteq$.*

Proof. Consider a passkey X and a subset $Y \subseteq X$. Suppose that Y is not a passkey, then there exists a passkey $Z : |Z| < |Y|$, $Z'' = Y''$. Hence, the set $Z \cup X \setminus Y$ is a key of X'', which is smaller in size than $Y \cup X \setminus Y = X$. This contradicts the fact that X is a passkey. □

This property is important since, given a passkey, we may conclude that all its subsets are passkeys. For example, knowing that acd is a passkey (see Fig. 3.3), we may conclude that all its proper subsets, namely ac, ad, cd, a, c, d are also passkeys.

3.4.4 Assessing data complexity

The closure index CI of a formal context allows us to estimate the number of elements in the corresponding concept lattice. The estimates are based on evaluating the most massive part of a concept lattice, which is usually related to a Boolean sublattice.

Definition 15. *A contranominal-scale context is a context of the form $(M, M, \neq)$, i.e., the context where a pair $x, y \in M$ belongs to the relation $(x, y) \in \neq$ iff $x \neq y$.*

The matrix of $(M, M, \neq)$ is filled with crosses except for the main diagonal. A contranominal-scale context represents an extreme case of context, where the concept lattice has $2^{|S|}$ elements, i.e., the maximal size, and each equivalence class is trivial, i.e., composed of the closed itemset itself. An example is given in Fig. 3.4.

Proposition 3 (lower bound on the lattice size)**.** *The closure index CI of $\mathbb{K} = (G, M, I)$ is a lower bound on the dimension of the inclusion-maximal Boolean sublattice of the concept lattice of (G, M, I), so the tight lower bound on the number of closed itemsets is 2^{CI}.*

Proof. In [AC17] (Lemma 6) it was proven that for a context $\mathbb{K} = (G, M, I)$ and $A \subseteq G$ there exists a contranominal-scale subcontext of $\mathbb{K}$ having A as its object set if and only if A is a minimal generator (key).

Following the definition of CI, the maximal size of passkeys in $\mathbb{K}$ is equal to CI, thus there is no passkey of larger size. Since the maximal size of a key is at least CI, the dimension of the maximal Boolean sublattice is at least CI.

Hence, the number of all closed itemsets $\mathcal{C}$ of (G, M, I) is at least 2^{CI}. □

	m_1	m_2	m_3	m_4
g_1		×	×	×
g_2	×		×	×
g_3	×	×		×
g_4	×	×	×	

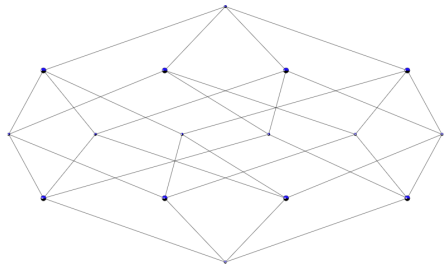


Figure 3.4: A contranominal-scale context of size 4 and the corresponding concept lattice

Note that the size of the maximal Boolean sublattice corresponds to the VC-dimension of the lattice (as a set family) [VC71]. Recall that VC-dimension of a set family is the maximal size of the set C such that its intersections with all elements of the family makes the powerset 2^C. Thus, the VC-dimension of the lattice is at least CI. However, it can be shown that in some cases the VC-dimension of the lattice can be larger than CI.

In many situations, a binarization is used for mining patterns in numerical data. We consider the *simplest binarization* where each interval of a pre-discretized attribute is mapped to a new binary attribute — so-called one-hot encoding — as it was explained in Section 2.2. In this case the closure index CI cannot be greater than the number of attributes in the original dataset. Below, we consider this extreme case, where CI is maximal.

Proposition 4 (complexity of binarized datasets)**.** *Let $\mathbb{K}_{num} = (G, S, W, I_{num})$ be a many-valued context and $\mathbb{K} = (G, M, I)$ be its simplest binarization. If the closure index CI of $\mathbb{K}$ is equal to the number of attributes $|S|$ in $\mathbb{K}_{num}$, then there exist CI objects in G such that their descriptions form a contranominal-scale subcontext of $\mathbb{K}$ and any pair of these descriptions in $\mathbb{K}_{num}$ differs in two attributes.*

Proof. Let us suppose that the number of original numerical attributes is equal to the closure index CI of $\mathbb{K}$. Thus, there exists a contranominal-scale context of size CI, where any pair of the corresponding objects have $CI-1$ matching binary values. Because of the bijection between the intervals of discretization and binary attributes, at least CI object descriptions differ in a specific pair of values. □

Example. The closure index CI of the formal context from Fig. 2.4 (middle) is equal to 2, i.e., the number of numerical attributes in the corresponding many-valued context (left). In the formal context, there exist several pairs of the objects that differ in two attributes, e.g., $\{g_1, g_2\}$, $\{g_1, g_3\}$, $\{g_1, g_4\}$.

3.4.5 Closeness under sampling

In many practical situations, datasets are large, and a way to efficiently analyze such datasets is to use data sampling, i.e., to build a model based on data fragments rather than on the whole dataset. There is a lot of approaches to sample data, depending on the task, data specificity and the selected learning models [BG09, BGG10, BLPG11, BMG12, DvLDR17].

In this section we consider a particular type of sampling that consists in selecting a certain amount of objects keeping the complete attribute space, i.e., we select rows but not columns.

Let $\mathbb{K} = (G, M, I)$ be a formal context. Then an *object sampling* in the context $\mathbb{K}$ is a context $\mathbb{K}_s = (G_s, M, I_s)$, where $G_s \subseteq G$, and $I_s = \{gIm \mid g \in G_s\}$.

Proposition 5. *Itemsets closed in $\mathbb{K}_s$ are closed in $\mathbb{K}$.*

Proof. Let $\mathbb{K} = (G, M, I)$ be a formal context, $\mathbb{K}_s = (G_s, M, I_s)$ be a formal subcontext, (A_s, B_s) be a formal concept of $\mathbb{K}_s$, B_s is closed in $\mathbb{K}_s$. We denote the derivation operators $(\cdot)'$ in context $\mathbb{K}$ by indexing them with the context name, i.e., $(\cdot)'_{\mathbb{K}}$.

Suppose that B_s is not closed in $\mathbb{K}$, i.e., there exists a concept (A, B) in $\mathbb{K}$ such that $B_s \subset B$ and $B_s \in Equiv(B)$ in $\mathbb{K}$. The later means that they have the same extent in $\mathbb{K}$, i.e., $(B)'_{\mathbb{K}} = (B_s)'_{\mathbb{K}}$. Since B_s is closed in $\mathbb{K}_s$ it follows directly that $B \notin Equiv(B_s)$, and from $B_s \subset B$ it follows that $(B)'_{\mathbb{K}_s} \subset (B_s)'_{\mathbb{K}_s}$.

$\mathcal{C}_1$	a	abc	c	d	ade	$abcdef$
		(b)			(e)	f
$\mathcal{C}_2$	ac		ad		cd	
$\mathcal{C}_3$			acd			

$\mathcal{C}_1$	a	b	ac	d	de	$bdef$
			(c)		(e)	(f)
$\mathcal{C}_2$	ad	ade	acd	abc	$abcdef$	
		ae	cd	(ab, bc)	(af, ce, cf)	

Figure 3.5: The closure structures computed on sampled contexts $\mathbb{K} = (G_s, M, I)$ with the sets $G_s = \{g_1, g_2, g_3, g_4\}$ (left) and $G_s = \{g_1, g_2, g_3, g_5\}$ (right). The passkeys that differ from their closure are highlighted in gray

Hence, $\exists g \in (B_s)'_{\mathbb{K}_s} \subseteq (B_s)'_{\mathbb{K}}$ such that $g \notin (B)'_{\mathbb{K}}$. This contradicts the fact that $B_s \in Equiv(B)$, thus our assumption is wrong, and B_s is closed in $\mathbb{K}$. □

Proposition 6. *For any closed itemset B, $Level(B)_{\mathbb{K}_s} \leq Level(B)_{\mathbb{K}}$, i.e., the passkey size of B for a sampled context is not greater than the passkey size of B for the whole context.*

Proof. Suppose the opposite, i.e., that $Level(B)_{\mathbb{K}_s} > Level(B)_{\mathbb{K}}$. Thus, there exists a passkey $X \in pKey(B)_{\mathbb{K}_s}$ and a passkey $Y \in pKey(B)_{\mathbb{K}}$ such that $|X| > |Y|$. The latter means that $(Y)''_{\mathbb{K}} = B$. However, from Proposition 2 and the definition of the passkey, for any $Y \subset B$, such that $|X| > |Y|$, $Y \notin Equiv(B)_{\mathbb{K}_s}$, thus, $(B)'_{\mathbb{K}_s} \subset (Y)'_{\mathbb{K}_s}$. It means that in the augmented context $\mathbb{K}$, $(B)'_{\mathbb{K}} \not\supseteq (Y)'_{\mathbb{K}}$, i.e., the extent of Y cannot be equal or included into extent of B, thus, $Y \notin Equiv(B)_{\mathbb{K}}$. Thus, there is no a passkey $Y \in pKey(B)_{\mathbb{K}}$, such that $|Y| < |X|$. Hence, our assumption is wrong and $Level(B)_{\mathbb{K}_s} \leq Level(B)_{\mathbb{K}}$. □

To illustrate that the closure level $Level(B)_{\mathbb{K}_s}$ of a concept (A, B) in a sampled context $\mathbb{K}_s$ is a lower bound of the closure level $Level(B)_{\mathbb{K}}$ of the concept in the larger context $\mathbb{K}$, we consider an example from Fig. 3.1. The closed itemset acd belongs to the 3rd level. Fig. 3.5 shows the closure structures for sampled context with objects $G_s = \{g_1, g_2, g_3, g_4\}$ (left) and $G_s = \{g_1, g_2, g_3, g_5\}$ (right). Closed itemset acd belongs to the 3rd and 2nd levels, respectively. Hence, a sampled context may be used to compute closed itemsets, and for any closed itemset B the following inequality holds: $Level(B)_{\mathbb{K}_s} \leq Level(B)_{\mathbb{K}}$.

3.5 The GDPM algorithm

In this section we introduce the GDPM algorithm — for Gradual Discovery in Pattern Mining — which computes the closure structure. We analyze the complexity of GDPM and then consider computational issues related to key-based structures. Finally we discuss how the closure structure may reduce the computational complexity of a mining problem.

3.5.1 Computing the closure structure with GDPM

The GDPM algorithm enumerates closed itemsets w.r.t. the closure levels based on lexicographically smallest passkeys. At each iteration GDPM computes the next closure level $\mathcal{C}_k$ using the itemsets from the current closure level $\mathcal{C}_{k-1}$. The algorithm can be stopped at any time. It returns the computed closure levels and, optionally, one passkey per closed itemset. GDPM requires that all attributes from M be linearly ordered, i.e., $m_1 < m_2 < \ldots < m_k$. Given this order, we say that an itemset $X_1 \subseteq M$ is *lexicographically smaller* than an itemset $X_2 \subseteq M$, if the smallest differing element belongs to X_1, or if X_1 is a prefix of X_2. For example, abf is lexicographically smaller than ac because $b < c$.

One of the main efficiency concerns in enumeration algorithms is to ensure the uniqueness of enumerated itemsets. To solve this issue, GDPM keeps all generated closed itemsets in a lexicographic tree to ensure that each closed itemset is generated only once. A *lexicographic tree* [Zak80] — *trie* or *prefix tree* — is a tree that has the following properties: (i) the root corresponds to the empty itemset; (ii) a node in the tree corresponds to an itemset, (iii) the parent of a node $i_1 \ldots i_{\ell-1} i_\ell$ is the node $i_1 \ldots i_{\ell-1}$, where the items are sorted lexicographically.

The pseudocode of GDPM is given in Algorithm 1. GDPM is a breadth-first search algorithm, which requires all attributes from M to be linearly ordered. The closure structure is computed sequentially level by level, a level $\mathcal{C}_k$ is computed by adding one by one attributes to each closed itemset from $\mathcal{C}_{k-1}$. A trie $\mathcal{T}_k$ is used to perform the duplicate test. In the beginning, a trie consists of the root labeled by the empty itemset. Then, at each iteration (line 4–7), the trie grows by including the closed itemsets from the closure level $\mathcal{C}_k$ that is currently being computed. The algorithm can be halted at any level.

Algorithm 1 GDPM (main loop)

Input: $\mathbb{K} = (G, M, I)$, a formal context, k_{max} the maximal closure level to compute
Output: $\mathcal{C}_k$, $k = 0, \ldots, k_{max}$, closed itemsets of the levels up to k_{max}

1: $k \leftarrow 0$
2: $\mathcal{C}_k \leftarrow \{\emptyset\}$
3: $\mathcal{T}_k \leftarrow \emptyset$
4: **repeat**
5: $\quad k \leftarrow k + 1$
6: $\quad \mathcal{C}_k \leftarrow GDPM\text{-}INT(\mathcal{C}_{k-1}, \mathcal{T}_{k-1}, \mathbb{K})$ // Algorithm 2
7: **until** $|\mathcal{C}_k| > 0$ and $k < k_{max}$
8: **return** $\mathcal{C}_k$

The function GDPM-INT for computing level $\mathcal{C}_k$ using itemsets from level $\mathcal{C}_{k-1}$ is described in Algorithm 2. The candidates for the closure level $\mathcal{C}_k$ are generated using a closed itemset B of the current closure level $\mathcal{C}_{k-1}$ and item m that is not included into B. For a candidate $B \cup \{m\}$ GDPM computes its closure (line 5), and then checks if this closed itemset is included in the trie (line 6). The closed itemsets that are not in the trie are added both to the trie $\mathcal{T}_k$ and the closure level $\mathcal{C}_k$ (line 7 and 8, respectively).

Algorithm 2 GDPM-INT

Input: $\mathcal{C}_{k-1}$, closed itemsets of the level $k-1$, $\mathcal{T}_{k-1}$, the trie containing all closed itemsets $\bigcup_{i=1,\ldots,k-1}\{B \mid B \in \mathcal{C}_i\}$
Output: $\mathcal{C}_k$, closed itemsets of the level k

1: $\mathcal{C}_k \leftarrow \emptyset$
2: $\mathcal{T}_k \leftarrow \mathcal{T}_{k-1}$
3: **for all** $B \in \mathcal{C}_{k-1}$ **do**
4: $\quad$ **for all** $m \in M \setminus B$ **do**
5: $\quad\quad B_c \leftarrow (B \cup \{m\})''$
6: $\quad\quad$ **if** $B_c \notin \mathcal{T}_k$ **then**
7: $\quad\quad\quad add(\mathcal{T}_k, B_c)$
8: $\quad\quad\quad add(\mathcal{C}_k, B_c)$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**
12: **return** $\mathcal{C}_k$

Algorithm 3 GDPM-EXT

Input: $\mathcal{C}_{k-1}$, closed itemsets of the level $k-1$, $\mathcal{T}_{k-1}$, the trie containing all closed itemsets $\bigcup_{i=1,\ldots,k-1}\{B \mid B \in \mathcal{C}_i\}$
Output: $\mathcal{C}_k$, closed itemsets of the level k

1: $\mathcal{C}_k \leftarrow \emptyset$
2: $\mathcal{T}_k \leftarrow \mathcal{T}_{k-1}$
3: **for all** $B \in \mathcal{C}_{k-1}$ **do**
4: $\quad$ **for all** $m \in M \setminus B$ **do**
5: $\quad\quad A_c \leftarrow (B \cup \{m\})'$
6: $\quad\quad$ **if** $A_c \notin \mathcal{T}_k$ **then**
7: $\quad\quad\quad add(\mathcal{T}_k, A_c)$
8: $\quad\quad\quad add(\mathcal{C}_k, A'_c)$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**
12: **return** $\mathcal{C}_k$

Example. Let us consider some intermediate steps of GDPM. We use the dataset from Fig. 3.1 and consider steps when a closed itemset $b \in \mathcal{C}_1$ is used to compute closed itemsets from $\mathcal{C}_2$. The result of the first call of GDPM-INT is $\mathcal{C}_1 = \{a, b, c, d, e, bef\}$. In the second call, after considering the candidates $\{a, m\}$ where $m \in \{b, c, d, e, f\}$, five new itemsets are added to the trie, namely abc, $abcdef$, ac, ad, ade. Then, the algorithm proceeds to compute the closed itemsets based on candidates $\{b, m\}$, $m \in \{c, d, e, f\}$. The execution tree of GDPM-INT and the trie are given in Fig. 3.6. We can see that for an itemset $\{b, m\}$, $m \in \{c, d, e, f\}$ GDPM-INT computes the closure $\{b, m\}''$ in line 5, and only after that checks the existence of $\{b, m\}''$ in the trie $\mathcal{T}_2$. The duplicate test $B_c \notin \mathcal{T}_2$ is performed in $O(|M|)$ steps.

Using the execution tree, we can also reconstruct the lexicographically smallest passkey. For example, itemset $bdef$ was generated in the branches labeled by "b" (to get a closed itemset b) and then "d". Thus,

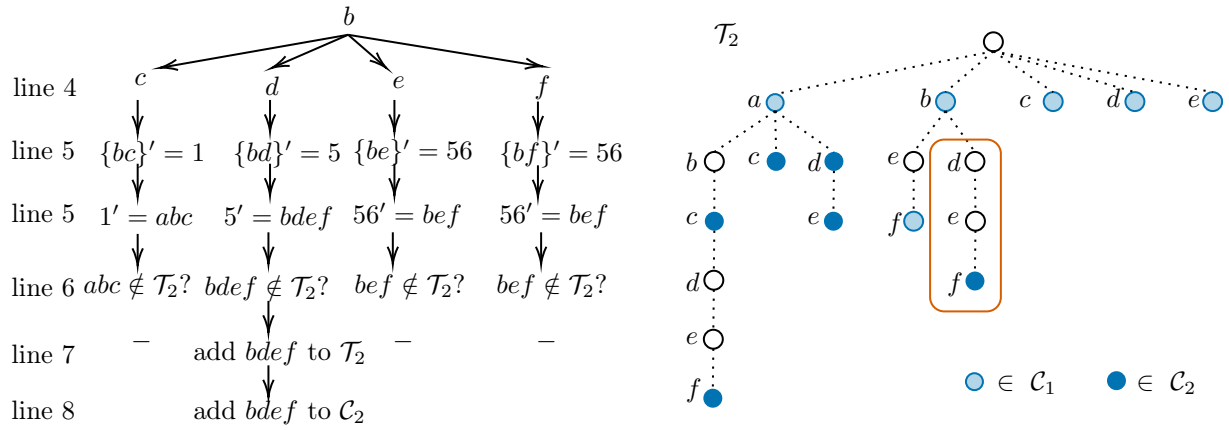


Figure 3.6: An iteration in the GDPM-INT algorithm for computing itemsets of the 2nd level using closed itemset b. The duplicate test (line 6) is performed after computing both the extent and intent (line 5). Elements which are newly added to $\mathcal{T}_2$ are highlighted in red

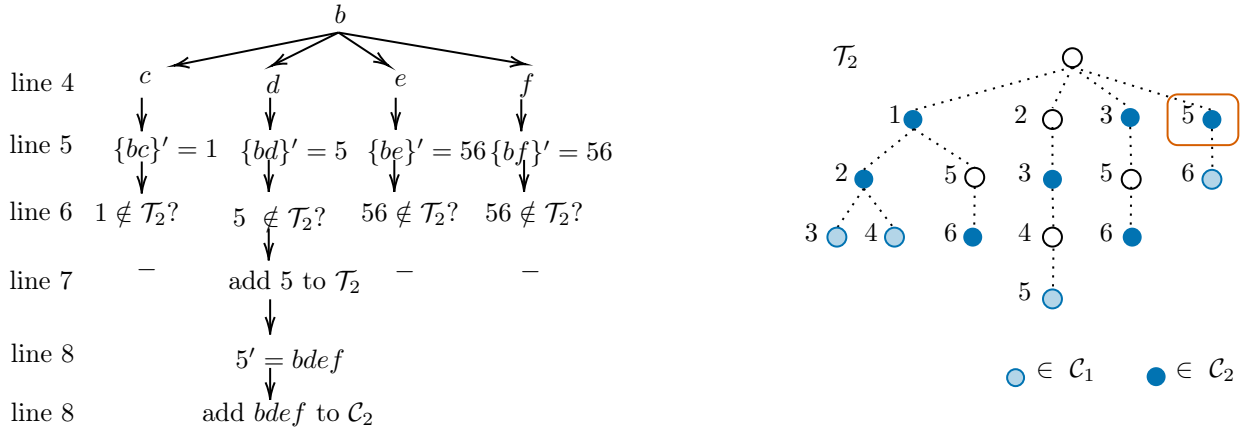


Figure 3.7: An iteration of the GDPM-EXT algorithm for computing itemsets of the 2nd level using closed itemset b. The duplicate test (line 6) is performed after computing an extent (line 5). The intent is computed only if the extent has not been yet created (line 8). An altered element in $\mathcal{T}_2$ is highlighted in red

the lexicographically smallest passkey of $bdef$ is bd.

3.5.2 The extent-based version of GDPM

In the previous section we considered the GDPM algorithm and the sub-algorithm GDPM-INT for computing a single closure level. To perform the duplicate test, GDPM-INT uses a trie based on closed itemsets (intents). Below we introduce an alternative version GDPM-EXT, where, instead of intents, the trie contains the extents of closed itemsets.

The difference is that instead of computing the closure $B_c = (B \cup \{m\})''$, GDPM-EXT computes only the extent $A_c = (B \cup \{m\})'$ (line 5). Then GDPM checks if A_c exists in the trie $\mathcal{T}_k$ (line 6) and inserts A_c into $\mathcal{T}_k$ if it is not found (line 7). For the inserted extent A_c, the corresponding closed itemset A'_c is added to the closure level $\mathcal{C}_k$ (line 8).

Example. Let us consider how GDPM-EXT works by means of a small example. We consider same steps as for GDPM-INT, i.e., when a closed itemset $b \in \mathcal{C}_1$ is used to compute closed itemsets from $\mathcal{C}_2$. The execution tree of GDPM-EXT and the trie are given in Fig. 3.7. Both GDPM-INT and GDPM-EXT generate closed itemsets in the same order, i.e., the result of the 1st call for both of them is $\mathcal{C}_1 = \{a, b, c, d, e, bef\}$. In the 2nd call after considering candidates $\{a, m\}$, $m \in \{b, c, d, e, f\}$ the following

new extents are added to the trie: 1, $\emptyset$, 12, 23, 3. The corresponding closed itemsets, added to $\mathcal{C}_k$, are the same as for GDPM-INT, i.e., abc, $abcdef$, ac, ad, ade. Then, GDPM-EXT proceeds to compute the closed itemsets based on $\{b, m\}$, $m \in \{c, d, e, f\}$. The execution tree in Fig. 3.7 shows that for an itemset $\{b, m\}$, $m \in \{c, d, e, f\}$ GDPM-EXT computes only $\{b, m\}'$ and then checks its existence in the trie $\mathcal{T}_2$. Comparing the execution trees in Fig. 3.6 and 3.7, we can see that the branches "c", "e", "f" of GDMP-EXT are shorter than the branches of GDPM-INT, thus GDPM-EXT may work faster than GDMP-INT. However, the duplicate test $X \in \mathcal{T}_2$ is performed in $O(|M|)$ and $O(|G|)$ steps for GDPM-INT and GDPM-EXT, respectively. Thus, in case where $|M| \ll |G|$, GDPM-INT may require smaller space than GDPM-EXT. Let us consider time and space complexity of GDMP in detail.

3.5.3 Complexity of GDPM

The estimates of the size of the next output. Given the input $\mathcal{C}_k$ (which is the output of the previous iteration), Algorithm 1 generates $|\mathcal{C}_{k+1}|$ concepts at the next iteration, the maximal number of the concepts is $\sum_{(A,B)\in\mathcal{C}_k} |M \setminus B|$. More generally, the number of concepts (or closed itemsets) at level $k + n$, $n \in \mathbb{N}$, is $O(|\mathcal{C}_k||M|^n)$.

The worst-case space complexity of GDPM. At each iteration Algorithm 1 stores the output of the previous iteration $\mathcal{C}_k$, the current output $\mathcal{C}_{k+1}$, and the trie with the closed itemsets/extents for GDPM-INT/EXT, respectively. Thus, at the $(k+1)$-th iteration GDPM requires $O\left((|\mathcal{C}_k| + |\mathcal{C}_{k+1}|) \cdot (|G| + |M|)\right)$ space to store the closed itemsets and $O(\sum_{i=1}^{k+1} |\mathcal{C}_i| \cdot |M|)$ or $O(\sum_{i=1}^{k+1} |\mathcal{C}_i| \cdot |G|)$ space to store the trie for GDPM-INT and GDPM-EXT, respectively. If the size of closed itemsets is smaller than the sizes of their extents, GDPM-INT may require less space than GDPM-EXT.

The worst-case time complexity of GDPM. To compute concepts $\mathcal{C}_{k+1}$ using $\mathcal{C}_k$ as output, one needs to check $O(|\mathcal{C}_k| \cdot |M|)$ candidates. For each candidate GDPM-INT computes both the extent and the intent, which takes $O(|G||M|)$. Then, the duplicate test takes $O(|M|)$. GDPM-EXT computes for each candidate only the extent, that takes $O(|G|)$ time, and then checks for duplicates in $O(|G|)$. Only for the candidates that pass the duplicate test it computes the intent taking $O(|G||M|)$ time. Despite the fact that dublicates are identified faster by GDPM-EXT, asymptotically computing a closed itemset by both algorithms takes $O(|G||M|)$. Hence, the total worst-case complexity of the $k + 1$-th iteration of GDPM-INT and GDPM-EXT is $O(|\mathcal{C}_k| \cdot |G| \cdot |M|^2)$.

3.5.4 Related approaches to key-based enumeration

In this section we consider related approaches to enumerating closed itemsets based on keys and the difference between them and GDPM.

As it was discussed in Section 3.2, one of the main efficiency concerns in enumeration algorithms is to ensure the uniqueness of enumerated closed itemsets (concepts). The canonicity test is known to be an efficient solution to this problem [KO02]. For performing the canonicity test the attributes from M are required to be linearly ordered (as for GDPM).

The canonicity test consists in checking if a newly computed closure of a generator is lexicographically greater than the generator itself. The canonicity test is applicable in depth-first search algorithms where itemsets are extended at each iteration by adding an item, thus the closed itemsets are enumerated in the "lexicographically smallest branch" of the execution tree. Unfortunately, we cannot use this approach when we generate closed itemsets based on keys and passkeys because the enumeration is based on passkeys which are not necessarily the lexicographically smallest elements in equivalence classes. Let us consider an example from Fig. 3.1 to illustrate this problem. The closed itemset ade has one key (which is a passkey) ae. Using the passkey ae, we get the closure ade that is lexicographically smaller than the passkey itself, thus, the canonicity test is not passed. Hence, no key and passkey can be used to generate ade relying on the canonisity test.

The keys (including passkeys) are computed by a breadth-first search algorithm in order to ensure enumeration of the keys before other generators. To the best of our knowledge, the first algorithm for

computing closed itemsets by enumerating the keys was proposed in [PBTL99b] and called A-Close. An alternative principle of enumeration was proposed in the Pascal algorithm [BTP+00] and improved in the Titanic algorithm [STB+02]. These three algorithms rely on the fact that keys make an order ideal, i.e., each subset of a key is a key (see Proposition 2).

In [SVNG09] a depth-first search algorithm, called Talky-G, for enumerating keys was proposed. This algorithm traverses the search space in so-called "reverse pre-order" and checks if a subset is a key by means of the hash table containing all enumerated previously keys. Another efficient approach for key enumeration called DefMe was proposed in [SR14].

The main challenge related to a *correct* enumeration of keys is to avoid enumeration of duplicates and non-keys. We call an enumeration algorithm *local* if it can eliminate a non-key itemset X by considering its proper subsets $X \setminus \{m\}$, $m \in X$, otherwise, it is called global.

The main advantage of the local enumeration is lower space requirements, since one needs to store only a subset of previously enumerated itemsets. Actually, to check if an itemset X is a key, one needs to check whether all its subsets of size $|X| - 1$ are keys.

The key can be enumerated locally using a breadth-first search approach. In this case, the keys of size k are generated at the k-th iteration. To check if a candidate is a key one needs to store only the output of the previous iteration, i.e., the keys of size $k - 1$. The depth-first approaches may work faster, but they require storage of all enumerated keys.

Let us consider the principles of key enumeration by a breadth-first algorithm called Titanic. The key enumeration is performed level by level and includes the following steps:

1. Candidate generation. The set of candidates $\mathcal{S}$ is generated by merging all keys from the previous level $\mathcal{K}_{k-1}$ that differ by one item: $\mathcal{S} = \{\{m_1 < \ldots < m_k\} \mid \{m_1, \ldots, m_{k-2}, m_{k-1}\}, \{m_1, \ldots, m_{k-2}, m_k\} \in \mathcal{K}_{k-1}\}$, where $\mathcal{K}_{k-1}$ is the set of all keys of size $k - 1$.

2. Verification of the ideal property, i.e., for each candidate one needs to check whether each of the k proper subsets of size $(k - 1)$ is included into $\mathcal{K}_{k-1}$.

3. Support-closedness verification, i.e., comparison of the actual support with the minimal support of keys from $\mathcal{K}_{k-1}$ contained in the current candidate.

The verification of keys (Step 2 and 3) is performed by considering k immediate subsets. The size of the largest possible level $\mathcal{K}_{|M|/2}$ is $O\left(\frac{2^{|M|+1/2}}{\sqrt{|M|\pi}}\right)$ (this follows directly from the Stirling's formula). Thus, the worst-case time complexity of the first step (consisting in considering all pairs of keys from $\mathcal{K}_{k-1}$ and comparing their elements) is $O\left(\frac{2^{2|M|+1}}{|M|\pi}|M|\right)$. However, this step can be optimized by storing all keys in a trie and merging each itemset corresponding to a key with its siblings that are lexicographically greater than the itemset. Completeness of this enumeration follows directly from the definition of the order ideal, i.e., any key of size k contains two keys of size $k - 1$ that differ only by the last elements. Thus, the worst-case time complexity of the optimized first step is $O\left(\frac{2^{|M|+1/2}}{\sqrt{|M|\pi}}|M|^2\right)$, because the multiplier $\frac{2^{|M|+1/2}}{\sqrt{|M|\pi}}$ is replaced by the maximal number of siblings $|M|$. This solution is also related to the problem of a "single parent" discussed in [AU09], where the authors proposed a depth-first search algorithm, called CloGenDFS, that traverses a search subspace (called a family tree) in such a way that each closed itemset has only one parent and thus is generated only one time. In the case of closed itemsets, it ensures the absence of duplicates. However, in the case of keys, using a breadth-first search traversal we may only expect a better execution time because each k-sized itemset will be generated only once, but to eliminate non-key elements, one should perform Steps 2 and 3.

Despite the locality of the Titanic algorithm, the worst-case space complexity is not polynomial and proportional to the size of the largest level, i.e., $O\left(\frac{2^{|M|+1/2}}{\sqrt{\pi|M|}}\right)$.

Even if the splitting induced by the sizes of keys is very similar to the closure structure (e.g., compare Fig. 3.2 and 3.3), there is an important difference in the strategies for ensuring the correct enumeration of

g_1	a	b	c	d
g_2		b	c	
g_3	a		c	
g_4	a	b		

$\mathcal{C}_1$	a	b	c	$abcd$
				(d)
$\mathcal{C}_2$	ab	bc	ac	

Figure 3.8: A dataset and the corresponding closure structure

keys and passkeys. The elements of the closure structure cannot be enumerated locally, since "minimality in size" cannot be verified by checking only immediate subsets.

We explain the difference by a small example from Fig. 3.8. To check, whether an itemset X is a key one needs to verify that all $|X|$ immediate itemsets of size $|X|-1$ are in this structure and belong to different equivalence classes than X. For example, when computing $\mathcal{C}_3$, to check whether abc is a key, it is enough to check whether that ab, bc, and ac are in $\mathcal{K}_2$ and belong to equivalence classes different from that of abc. For passkeys one also needs to check "minimality in size" in the equivalence class. Key abc does not comply with the last constraint, since the smallest passkey is $d \in Equiv(abc'')$. To verify it one should check the passkeys in $\mathcal{K}_1$. Thus, for computing the closure structure one needs to store all closed itemsets corresponding to the enumerated passkeys, and the worst-case space complexity of the algorithm is proportional to the number of closed itemsets $O(2^{|M|})$. However, the related problem — enumeration of the passkeys — may require much less space than enumeration of keys, since the number of keys can be exponentially larger than the number of passkeys (see Section 3.5.5, Proposition 8).

3.5.5 Computing passkeys. Towards polynomial complexity

The complexity of the GDPM algorithm is higher than that of simple enumerators of closed itemsets. However, knowing the closure level to which a closed itemset belongs allows us to significantly reduce the complexity of enumeration of its passkeys.

Keys play an important role in FCA and in the related fields [HPY00], e.g., graph theory, database theory, etc. Below we list the complexity of some problems related to computing or enumerating keys or passkeys. It is known that computing keys and generators of a closed itemset is intractable. In [Kuz07], it was shown that the problem of enumerating keys is #P-complete[3]. The authors of [GKM+03] showed that the problem of enumerating keys is #P-complete. More precisely, they studied the problem of enumerating minimal keys of a relational system. Because of the correspondence between minimal keys of a relational system and keys — in the sense we use in this paper — one can use this result to compute the complexity of enumerating keys. Finally, in [HS08] the authors studied the problem of enumerating passkeys and showed that this problem is #·coNP-complete.

Let us consider the complexity of the problem of computing the passkeys.

Problem 1. *PASSKEY*
INPUT: context $\mathbb{K} = (G, M, I)$ and subset of attributes $X \subseteq M$.
QUESTION: Is X a passkey, i.e., it is a key of X'' and there is no key of X'' of size less than $|X|$?

Proposition 7. *The PASSKEY problem is coNP-complete.*

Proof. We introduce the following auxiliary problems:

Problem 2. *KEY OF SIZE AT MOST K*
INPUT: context $\mathbb{K} = (G, M, I)$, intent $B \subseteq M$ and natural number $k \in \mathbb{N}$.
QUESTION: Is there a key of B of size less than or equal to k?

[3]In particular, see Theorem 4, where the numerator of integral stability index is the number of generators of a closed itemset.

Problem 3. *NO KEY OF SIZE AT MOST K*
INPUT: context $\mathbb{K} = (G, M, I)$*, intent* $B \subseteq M$ *and natural number* $k \in \mathbb{N}$*.*
QUESTION: Is it true that there is no key of B *of size less than or equal to* k*?*

Problem 2 is trivially polynomially equivalent to the famous NP-complete SET COVERING problem. Thus, Problem 3 is coNP-complete, and this problem is polynomially reducible to PASSKEY, so PASSKEY is coNP-complete. □

Corollary. Passkeys cannot be enumerated in total polynomial time unless P = coNP.

Thus, enumeration of the keys or generators are computationally hard. However, knowing the closure level to which a closed itemset belongs, the problem of enumerating passkeys becomes polynomial since for a closed itemset $B \in \mathcal{C}_k$, the number of all possible passkeys is given by

$$\binom{|M|}{k} = \frac{|M|!}{k!(|M|-k)!} = \frac{(|M|-k+1)\cdot\ldots\cdot|M|}{k!} \approx \frac{|M|^k}{k!} \in O(|M|^k).$$

Thus, if k is constant, one can identify all passkeys in polynomial time. We also emphasize that the maximal possible k depends on M, i.e., $k \leq |M|$, however for the concepts from the first levels, where k is small, we can neglect the fact that k may be proportional to M. Moreover, a closed itemset B from the k-th level provides the implications with the shortest antecedent k, i.e., size of the passkey, and largest consequent $|B| - k$. Thus the first levels provide the implications that are commonly considered to be the best for interpretation purposes [BTP+00].

Moreover, passkeys may provide much shorter representation of equivalence classes than keys.

Proposition 8. *Consider the context* $\mathbb{K} = (G, M, I)$*, where* $G = \{g_0, \ldots, g_n\}$*,* $M = \{m_0, \ldots, m_{2n}\}$*,* $g_0' = M$*, and* $g_i' = M \setminus \{m_0, m_i, m_{n+i}\}$*. The number of keys is* 2^n*, whereas there is a unique passkey.*

An example of such a context for $n = 3$ is given in Fig. 3.9. The elements of the equivalence class of formal concept (g_0, M), which has a unique passkey and exponentially many keys, are given on the right.

	m_0	m_1	m_2	m_3	m_4	m_5	m_6
g_0	×	×	×	×	×	×	×
g_1			×	×		×	×
g_2		×		×	×		×
g_3		×	×		×	×	

set name	members of the set
$pKey(M)$	m_0
$Key(M) \setminus pKey(M)$	$m_1m_2m_3$, $m_1m_2m_6$, $m_1m_5m_3$, $m_1m_5m_6$, $m_4m_2m_3$, $m_4m_2m_6$, $m_4m_5m_6$, $m_4m_5m_6$
$Equiv(M) \setminus Key(M)$	$m_0m_1m_2m_3$, $m_0m_1m_2m_6$, $m_0m_1m_5m_3$, $m_0m_1m_5m_6$, $m_0m_4m_2m_3$, $m_0m_4m_2m_6$, $m_0m_4m_5m_6$, $m_0m_4m_5m_6$

Figure 3.9: An example of the context with the closed itemset M having exponentially many keys and a unique passkey

Proof. Let us consider the number of combinations of attributes that make a contranominal-scale subcontext of size n. The set of objects of this subcontext is $\{g_1, \ldots, g_n\}$. Each attribute with the extent $\{g_1, \ldots, g_n\} \setminus \{g_i\}$ can be chosen in 2 different ways, i.e., either m_i or m_{i+n}. Thus, there exist 2^n ways to create the contranominal-scale subcontexts of size n with attribute extents $\{g_1, g_2, \ldots, g_n\} \setminus \{g_i\}$, $i = 1, \ldots, n$. All these 2^n attribute combinations have the same extent $\{g_0\}$ in the original context $\mathbb{K}$. In this context $\mathbb{K}$, there is another attribute m_0 having the same extent $\{g_0\}$. It means that the equivalence class $Equiv(M)$ has $2^n + 1$ keys, namely, $\{m_0\} \cup \{m_{1+k_1}m_{2+k_2}\ldots m_{n+k_n} \mid k_i \in \{0, n\}, i = 1, \ldots, n\}$, and only one passkey m_0. □

Table 3.2: Parameters of the datasets

name	$\|G\|$	$\|M\|$	density	#classes	#attr.	closure index CI	$2^{CI} \leq \|\mathcal{C}\| \leq 2^{\min(\|M\|,\|G\|)}$
adult	48842	95	0.15	2	14	12	4.1e+3 $\leq$ 3.6e+5 $\leq$ 4.0e+28
auto	205	129	0.19	6	**8**	**8**	2.6e+2 $\leq$ 5.8e+4 $\leq$ 6.8e+38
breast	699	14	0.64	2	10	6	6.4e+1 $\leq$ 3.6e+2 $\leq$ 1.6e+4
car evaluation	1728	21	0.29	4	**6**	**6**	6.4e+1 $\leq$ 8.0e+3 $\leq$ 2.1e+6
chess kr k	28056	40	0.15	18	**6**	**6**	6.4e+1 $\leq$ 8.5e+4 $\leq$ 1.1e+12
cylinder bands	540	120	0.28	2	39	19	5.2e+5 $\leq$ 4.0e+7 $\leq$ 1.3e+36
dermatology	366	43	0.28	6	33	9	5.1e+2 $\leq$ 1.4e+4 $\leq$ 8.8e+12
ecoli	327	24	0.29	5	8	6	6.4e+1 $\leq$ 4.3e+2 $\leq$ 1.7e+7
glass	214	40	0.22	6	10	8	2.6e+2 $\leq$ 3.2e+3 $\leq$ 1.1e+12
heart-disease	303	45	0.29	5		9	5.1e+2 $\leq$ 2.6e+4 $\leq$ 3.5e+13
hepatitis	155	50	0.36	2	19	11	2.0e+3 $\leq$ 1.4e+5 $\leq$ 1.1e+15
horse colic	368	81	0.21	2	27	9	5.1e+2 $\leq$ 1.7e+5 $\leq$ 2.4e+24
ionosphere	351	155	0.22	2	34	17	1.3e+5 $\leq$ 2.3e+7 $\leq$ 4.6e+46
iris	150	16	0.25	3	**4**	**4**	1.6e+1 $\leq$ 1.2e+2 $\leq$ 6.6e+4
led7	3200	14	0.50	10	**7**	**7**	1.3e+2 $\leq$ 2.0e+3 $\leq$ 1.6e+4
mushroom	8124	88	0.25	2	22	10	1.0e+3 $\leq$ 1.8e+5 $\leq$ 3.1e+26
nursery	12960	27	0.30	5	**8**	**8**	2.6e+2 $\leq$ 1.2e+5 $\leq$ 1.3e+8
page blocks	5473	39	0.26	5	10	8	2.6e+2 $\leq$ 7.2e+2 $\leq$ 5.5e+11
pen digits	10992	76	0.21	10	16	11	2.0e+3 $\leq$ 3.6e+6 $\leq$ 7.6e+22
pima	768	36	0.22	2		8	2.6e+2 $\leq$ 1.6e+3 $\leq$ 6.9e+10
soybean	683	99	0.32	19	35	13	8.2e+3 $\leq$ 2.9e+6 $\leq$ 6.3e+29
tic tac toe	958	27	0.33	2	9	7	1.3e+2 $\leq$ 4.3e+4 $\leq$ 1.3e+8
wine	178	65	0.20	3	13	7	1.3e+2 $\leq$ 1.3e+4 $\leq$ 3.7e+19
zoo	101	35	0.46	7	17	7	1.3e+2 $\leq$ 4.6e+3 $\leq$ 3.4e+10

3.6 Experiments

3.6.1 Characteristics of the datasets

In this section we study some properties of the closure structure on real-world datasets. We use datasets from the LUCS-KDD repository [Coe03], their basic statistics are given in Table 3.2. The number of attributes in original/binarized data is given in columns "#attr." and "$|M|$", respectively. The relative number of "1"s in binary data is given in column "density". We do not use the class labels of objects to compute the closure structure, but we use them to evaluate itemsets in Section 3.6.5. The number of classes is given in column "#classes".

First of all, we study the closure levels of data and show how they are related to the size of the datasets and the estimated number of concepts. Column CI contains the total number of closure levels. These values are quite low w.r.t. the total number of attributes $|M|$ but usually close to the number of attributes in the original data (i.e., "#attributes"). In our experiments we observe several datasets, where the closure index CI is equal to the number of attributes in the original data, namely "auto", "car evaluation", "chess kr k", "iris", "led7", "nursery". Thus, from Proposition 4, there exist at least CI object descriptions such that each pair of them differ in only one pair of numerical value (where numerical values are distinguished up to binarization intervals). For "car evaluation" and "nursery" the number of closed itemsets of the highest complexity is equal to the number of objects, i.e., for each object description and each of its numerical value there exists another object description that differs from it in a pair of attributes.

The observed variability of attribute values may be caused by a very fine-grained (too detailed) discretization. This variability may hamper itemset mining and can serve as an indicator for choosing a

coarser binarization. In this paper we do not study an appropriate binarization strategy, but use the one generally used in related studies [Coe03].

However, high values of CI may be caused not only by a too fine-grained discretization of real-valued attributes, but also the "probabilistic" nature of data, i.e., when the relation between object g and attribute m may appear with a certain probability, e.g., in the market basket analysis an item (dis)appears in a transaction with a certain probability. We consider this kind of data in Section 3.6.6.

In Section 3.4.4 we mentioned that CI may be used to compute a lower bound on the size of the concept lattice, i.e., the total number of concepts/closed itemsets is bounded from below by 2^{CI}. In the last column of Table 3.2 we list this lower bound together with the actual number of closed itemsets $\mathcal{C}$ and the upper bound $2^{\min(|M|,|G|)}$. Our experiments clearly show that the lower bound is much closer than the upper bound to the actual value of the lattice size.

In this section we considered the size of the closure structure w.r.t. the dataset size and discussed some indicators of noise or too detailed discretization. In the next section we consider computational aspects of the closure structures and discuss the performance of the GDPM algorithm[4].

3.6.2 Computational performance

As it was mentioned in Section 3.5.4, when using passkeys to generate closed itemsets, we cannot rely on the canonicity test to ensure uniqueness in generating closed itemsets. Thus, the main limitations of the GDPM algorithm are that (i) it may take a lot of space for a trie to store all generated closed itemsets, (ii) the traversal strategy may also be redundant because we cannot use an efficient strategy to check uniqueness of the generated closed itemset. To estimate the impact of these limitations we study the execution time and the number of nodes in the tries. The results are reported in Table 3.3.

Our experiments show that for small datasets, i.e., containing less than or about 1K objects and a small number of attributes, the difference between the algorithms is negligible. For larger datasets GDPM-EXT works faster when the number of attributes in original and binarized dataset is high and the number of objects is not too large. However, both versions are not scalable, meaning that as the size of datasets increases, the computational time increases considerably. We compare GDPM with the closest combination of the algorithms "Charm+Talky-G" implemented in the Coron data mining platform[5]. The Charm algorithm [ZH99] enumerates closed itemsets while Talky-G [SVN+14] enumerates all keys. The experiments show that GDPM works slower than the closest in functionality combination of algorithms. Actually, these results leave space for further study and improvement of the GDPM algorithm, in particular, by using hash tables and vertical representation of datasets. It is also important to notice that GDPM and "Charm+Talky-G" perform different tasks.

Below we consider more deeply the behavior of algorithms GDPM-EXT and GDPM-INT in comparing the execution time and the size of the tries for the datasets where we can observe the difference between the algorithms. The level-wise execution time is reported in Fig. 3.10. As it can be seen, the first and last levels are fast to compute. The most time-consuming levels are located in the middle of the closure structure. For example, for "adult" levels 1–3 and 9–12 are computed in no more than 10 seconds. While computing level 5 takes 319 and 233 seconds for GDPM-INT and GDPM-EXT, respectively. Almost for all datasets GDPM-EXT works faster than GDPM-INT. We observe the largest difference in the running time for "pen digits", where computing level 6 took 4855 and 1222 seconds for GDPM-INT and GDPM-EXT, respectively. However, for some datasets GDPM-INT works faster, e.g., for "cylinder bands" computing level 10 takes 874 seconds for the GDPM-EXT algorithm and only 316 seconds for GDPM-INT. Moreover, the running time curve of GDPM-INT is flatter and does not have a pronounced peak.

The trie that stores the generating closed itemsets grows at each iteration. Fig. 3.11 shows how the tries are growing with each new closure level. We also show the evolution of the size of tries for the datasets where this difference is most distinguishable, namely "led7", "pima" and "zoo". For the remaining datasets the sizes of tries that store closed itemsets and their extents are quite close one to another one. The total sizes of tries, as well as the corresponding running times for all datasets are reported in Table 3.3.

[4]The source code of GDPM is available at `https://github.com/makhalova/GDPM`

[5]`http://coron.loria.fr/`

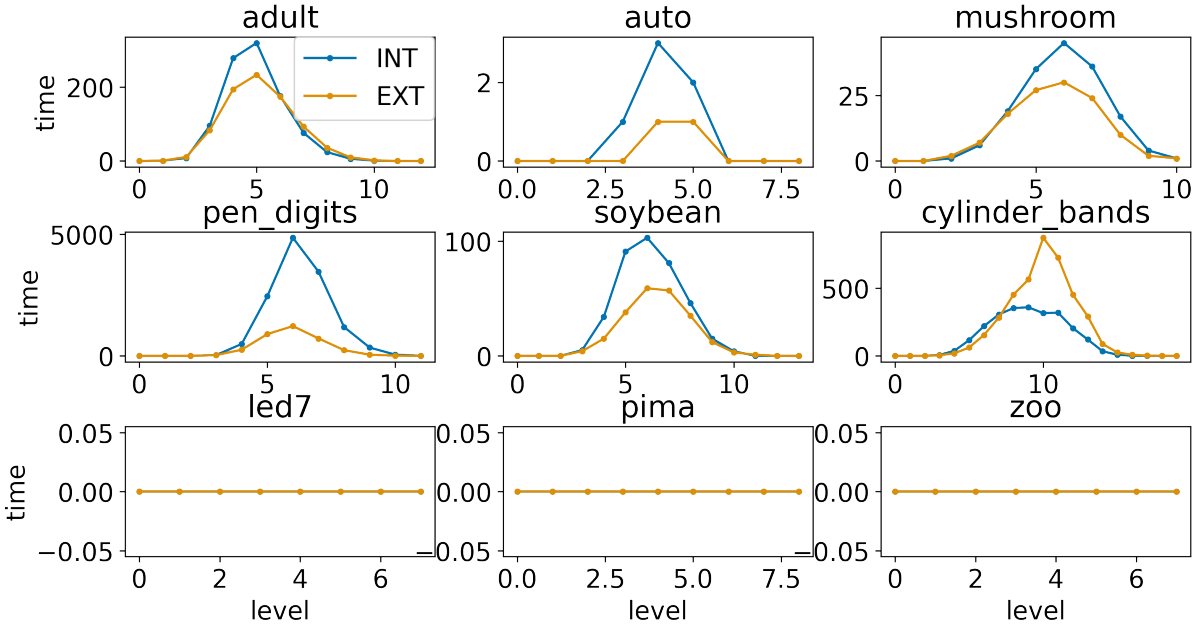


Figure 3.10: Level-wise running time of GDPM-INT / EXT for some datasets, the level number and time in seconds are given in axes x and y, respectively. Level 0 contains only itemset $\emptyset$ in the root

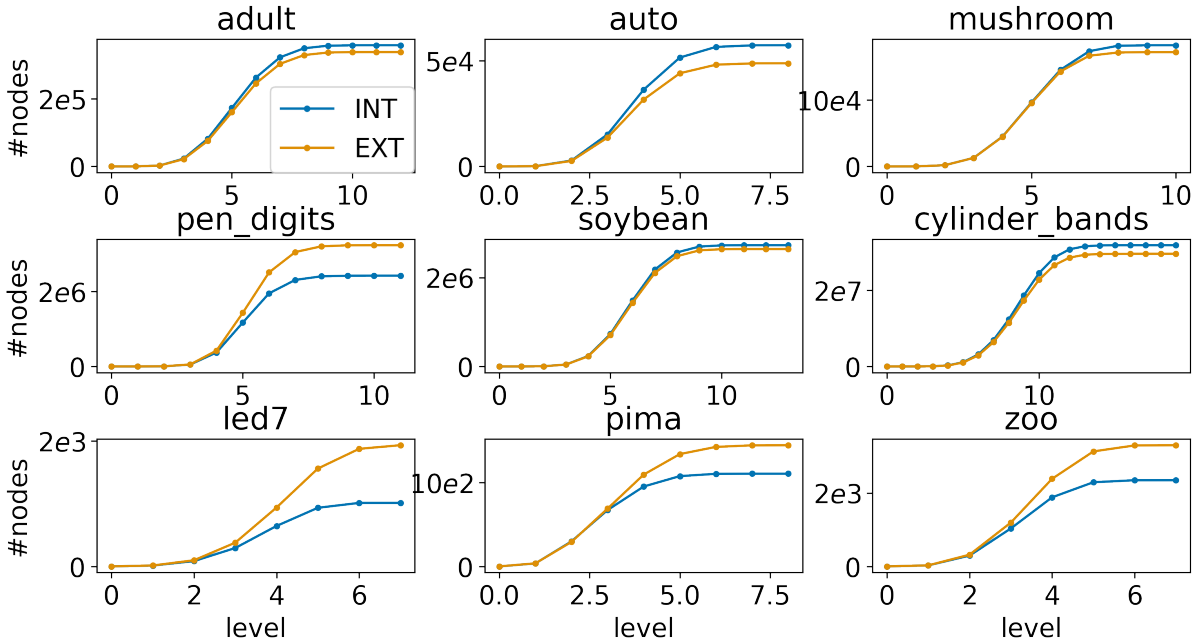


Figure 3.11: Growing of the tries in the number of nodes, the level number, and the trie size in nodes are given in axes x and y, respectively

Table 3.3: Performance of GDPM-INT/EXT and key enumerator TALKY-G

name	$\|G\|$	$\|M\|$	#attr.	#closed itemsets	#nodes in $\mathcal{T}$		runtime, sec		
					INT	EXT	INT	EXT	Charm + Talky-G
adult	48842	95	14	359141	359141	338906	984	837	13
auto	205	129	8	57789	57390	48834	6	2	5
breast	699	14	10	361	361	354	0	0	1
car evaluation	1728	21	6	8000	4875	6901	1	0	1
chess kr k	28056	40	6	84636	54761	68869	146	148	5
cylinder bands	540	120	39	39829537	39829537	37007525	2404	4010	-*
dermatology	366	43	33	14152	10506	12172	1	1	1
ecoli	327	24	8	425	425	373	0	0	1
glass	214	40	10	3246	2393	2757	0	0	1
heart-disease	303	45	0	25539	19388	21869	1	1	1
hepatitis	155	50	19	144871	88039	122277	2	1	5
horse colic	368	81	27	173866	138075	138075	11	9	10
ionosphere	351	155	34	23202541	23202541	20354049	2467	2585	12628
iris	150	16	4	120	80	111	0	0	1
led7	3200	14	7	1951	1012	1931	0	0	1
mushroom	8124	88	22	181945	181945	171590	164	121	8
nursery	12960	27	8	115200	76800	103351	46	51	4
page blocks	5473	39	10	723	451	702	0	3	1
pen digits	10992	76	16	3605507	2423123	3236109	12863	3398	123
pima	768	36	0	1626	1105	1444	0	0	1
soybean	683	99	35	2874252	2726204	2640461	379	224	298
tic tac toe	958	27	9	42712	25276	34513	1	1	1
wine	178	65	13	13229	8518	10202	1	1	2
zoo	101	35	17	4570	2933	4125	0	0	1
average	5239	57	15	2947747	2883953	2680313	812	475	376

* The algorithm crashes because of the lack of memory.

We emphasize that the size of the trie does not affect a lot the runtime of the duplicate test, i.e., checking whether a closed itemset is generated for the first time, which takes at most $O(|M|)$ and $O(|G|)$ time for GDPM-INT and GDPM-EXT, respectively.

The analysis of the algorithms shows that both algorithms compute first levels fast using not much space. However, it remains to study whether we can stop computation at the first levels and whether the itemsets from the first levels are useful. We address these questions in the next sections.

3.6.3 Data topology or frequency distribution within levels

The common approach to frequent itemset mining consists in gradually mining itemsets of decreasing frequency. For frequency-based enumeration methods infrequent itemsets are the last to be computed. As we discussed above, it results in the need for computing exponentially many itemsets. However, not all frequent itemsets are interesting and there are not that many infrequent itemsets that are worth considering.

In this section we study the frequency distribution of itemsets within closure levels. We take the number of itemsets at level k as 100% and compute the ratio of itemsets within 5 frequency bins: [0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0]. In Fig. 3.12 we report the distribution within these frequency bins. For example, the figure shows that "breast" dataset has 6 levels, the largest one is the 4th, since it

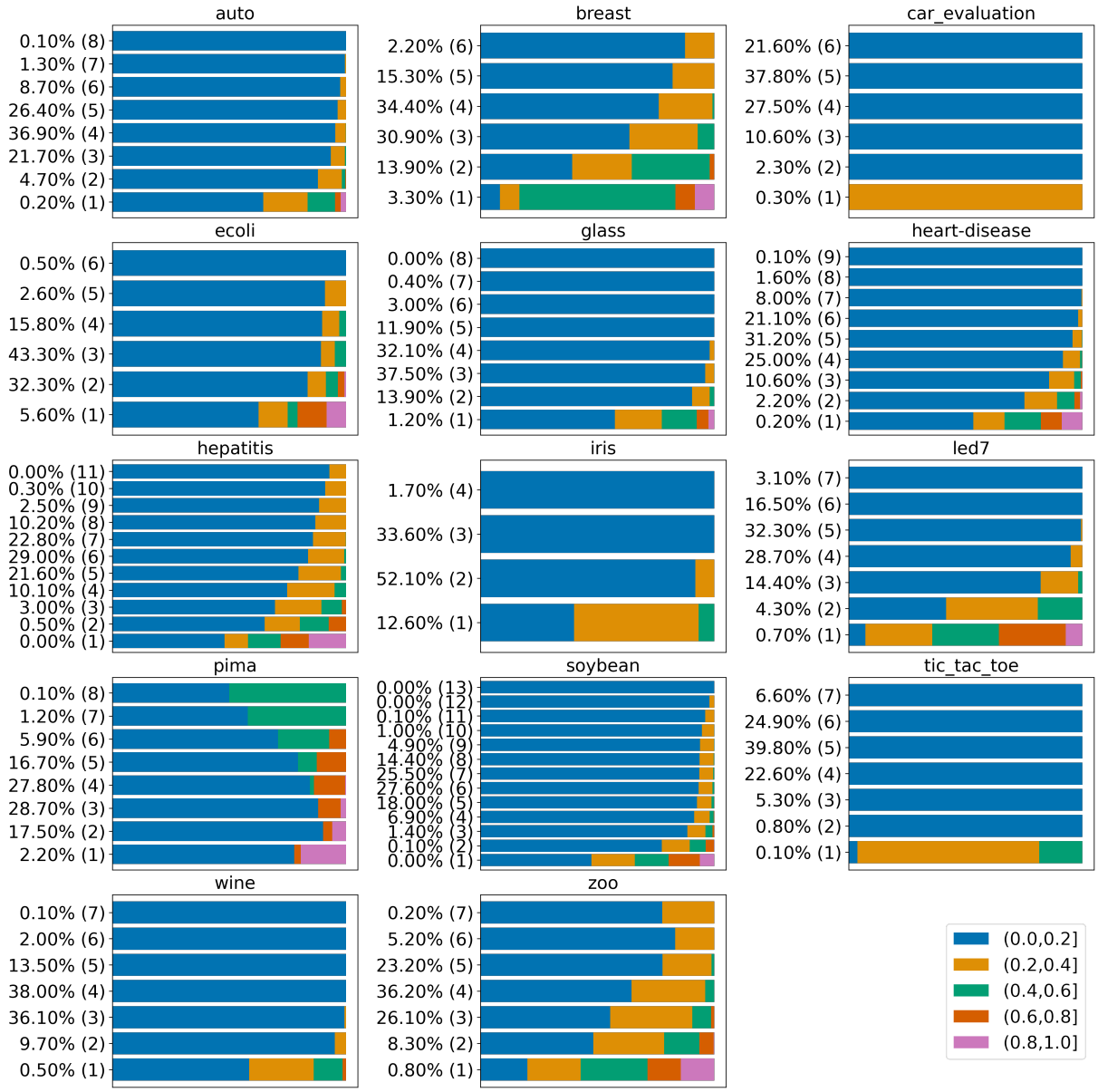


Figure 3.12: Distribution of closed itemsets within 5 frequency bins by the closure levels. The horizontal bars represent closure levels, the level number is given in parentheses. The rightmost value is the percentage of closed itemsets $|\mathcal{C}_k|/|\mathcal{C}|$. The width of the bar is proportional to the ratio of $\mathcal{C}_k$ of frequency $(v_1, v_2]$ among $\mathcal{C}_k$

accounts for 34.4% of all closed itemsets. The first level contains itemsets of all frequency bins, while the last two levels only contain itemsets of frequency at most 0.4, i.e., [0, 0.2) and [0.2, 0.4).

Almost in all datasets we observe a quite similar behavior: the proportion of frequent itemsets decreases with the closure levels. Actually, the first level contains a relatively small ratio of infrequent itemsets and a quite large ratio of frequent ones. In the next levels the ratio of frequent itemsets decreases, while the ratio of infrequent ones increases.

For example, for "breast" dataset, closed itemsets of frequency in $(0.8, 1.0]$ (violet) are present only in the 1st level, where they constitute 8.3% of all closed itemsets of $\mathcal{C}_1$. The closed itemsets of frequency in $(0.6, 0.8]$ (red) account for 8.3% of $\mathcal{C}_1$ and 2% of $\mathcal{C}_2$. The closed itemsets of frequency $(0.4, 0.6]$ (green)

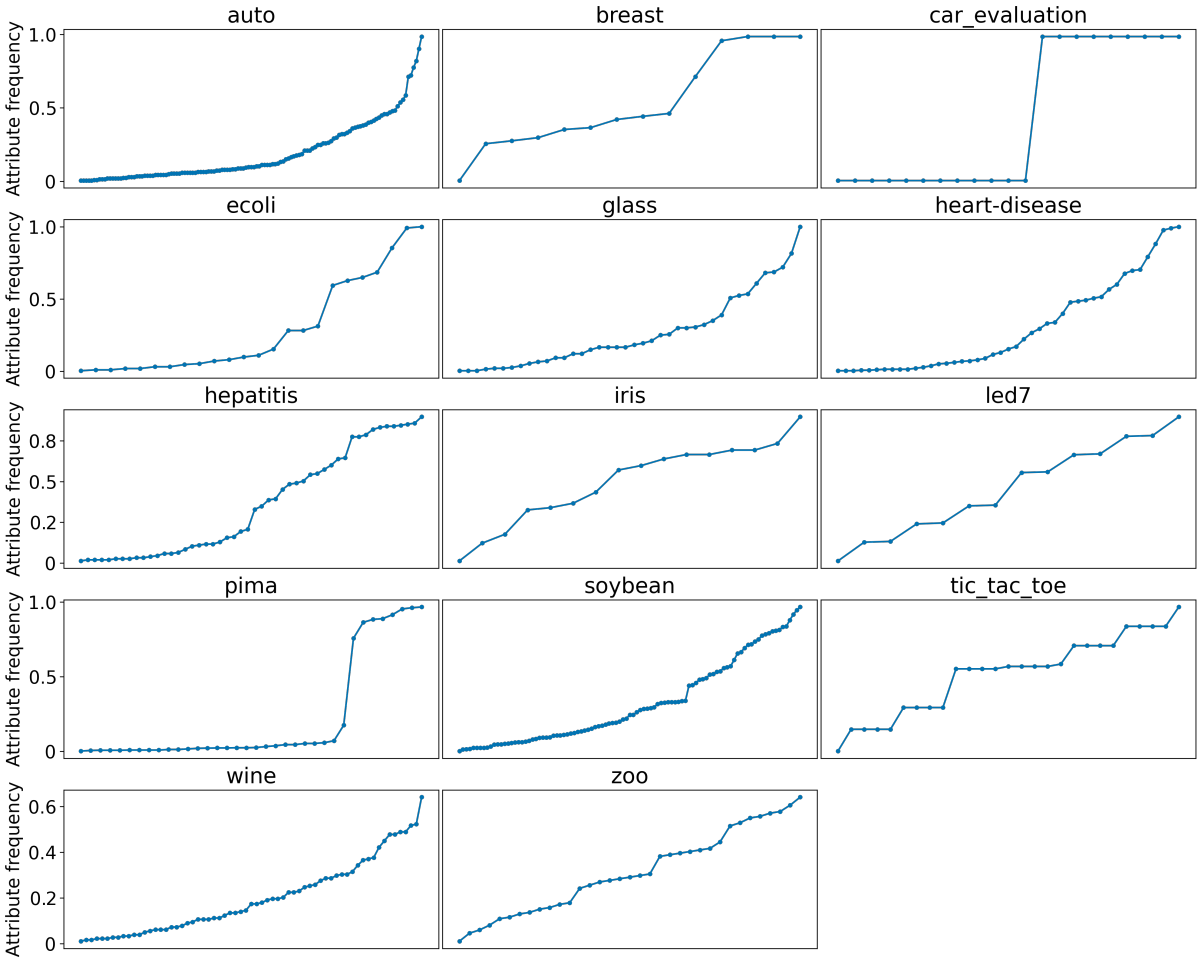


Figure 3.13: Frequency of attributes in ascending order for some datasets

account for 67% of $\mathcal{C}_1$, 33% of $\mathcal{C}_2$, 7.1% of $\mathcal{C}_3$, and 0.8% of $\mathcal{C}_4$. The less frequent ones (frequency below 0.2) constitute 8.3% of $\mathcal{C}_1$ and 87.5% of the last level $\mathcal{C}_6$.

However, some datasets, e.g., "car evaluation" and "pima," show a different behaviour: the rate of frequent itemsets increases with the closure level. This is typical for the datasets that have two types of attributes, e.g., of very low and very high frequency. To illustrate it, we show in Fig. 3.13 the attribute frequency for the datasets given in Fig. 3.12. We can see that the attribute frequency distributions of "page blocks" and "pima" differ from the others, i.e., there exist two groups of attributes of very contrasted frequencies. Thus, the frequency distribution of attributes impacts the itemset frequency distributions within the closure levels.

For the majority of datasets, the first closure level contains itemsets of diverse frequencies, thus by generating only a *polynomial* number of itemsets we may obtain itemsets of quite different frequencies.

Moreover, the frequency distributions by levels, as shown in Fig. 3.12, provide a representation of the "topology" of the dataset in terms of closed sets. For example, for "breast" dataset we may infer that we need at most 6 items to describe any closed itemset. Moreover, we may use only itemsets of size 1 or 2 to concisely describe itemset of frequency above 0.6. The closure structure provides the number of closed itemsets of a given frequency that can be represented without loss by passkeys of size k. For example, the first four levels of "breast" dataset contain 8 (66.7%), 17 (33.3%), 8 (7.1%) and 1 (0.7%) closed itemsets of frequency $(0.4, 0.6]$. Thus the majority of these frequent closed itemsets can be succinctly represented by passkeys of size 2 or 3.

From the experiments we can see that the first levels contain the itemsets with the most "diverse"

frequency ranges. However, it is not clear how these itemsets cover the data in a satisfying way and this is what we study in the next section.

3.6.4 Coverage and overlaps

Usually, we want that the generated concepts cover data in the most uniform way, i.e., that the whole dataset is completely covered and that each entry is covered by roughly the same number of itemsets. We evaluate the level-wise distribution of the closed itemsets by measuring the coverage and overlap ratios.

Let $\mathbb{K} = (G, M, I)$ be a formal context and $\mathcal{C}$ be an arbitrary set of formal concepts, then the coverage ratio is defined as the ratio of non-empty entries in the context covered by the concepts from $\mathcal{C}$. The overlap ratio is defined as the average number of itemsets that cover a cell in the data. Since the overlap ratio is computed for the entries covered by at least one itemset, it may take any value from 1 to the number of itemsets.

Fig. 3.14 shows the coverage rate and overlap ratios by levels for some datasets. The itemsets from the first level always cover the whole dataset. Then, the coverage rate steadily decreases with each new level. For example, the coverage rates of the first levels of "pima" dataset are close to 1, at level 5 the coverage rate drops to 0.98, at level 7 it decreases to 0.82, and at level 8 it drops to 0.56.

The overlap ratio — the average number of itemsets that cover an entry of the dataset — may vary a lot within a single level, but usually the closure levels in the middle have the largest average number of itemsets per entry. For example, the overlap ratio of "pima" may vary a lot from level to level. The overlap ratio at the 2nd level is low (equals to 7.32 ± 1.24), at the next levels it increases and achieves its maximum at the level 4 (30.27 ± 8.09). The latter means that each entry of a dataset is covered on average by 30 itemsets from the level 4.

Almost for all datasets we observe quite the same behavior: usually the coverage rate starts to decrease at the levels having the highest overlap ratio. We also observe that with quite the same overlap ratio the itemsets from the first levels (before the peak on the overlap curve) cover much larger fragments of data than those from the last levels. The latter means that, compared with the first levels, the description of the last levels is more focused on very particular data fragments.

However, based on these experiments, we should now investigate whether the itemsets from the first levels are more useful than those from the last levels.

3.6.5 Usefulness of concepts

In the previous sections we considered the closure levels and showed that the first levels contain itemsets with more diverse frequencies, with a larger amount of frequent itemsets and a smaller number of infrequent ones. In Section 3.4.2 we also mentioned that the itemsets from the first levels can be replaced by passkeys from the same equivalence class, allowing for much more succinct descriptions.

In this section we study the usefulness of the generated closed itemsets and focus on the ability of itemsets to describe "actual" classes in data. To measure this ability of an itemset we study the average F1 measure by levels. We consider each closed itemset B with its extent A as well as the class labels that were not used before. For a formal concept (A, B), where each object $g \in A$ has an associated class label $class(g) \in \mathcal{Y}$, the class label of an itemset is computed by the majority vote scheme, i.e., $class(B) = \arg\max_{y \in \mathcal{Y}} \{|\{class(g) = y\}| \mid g \in A\}$. Then, F1 measure for (A, B) is given by

$$F1(A, B) = \frac{2tp}{2tp + fp + fn}$$

where $tp = |\{g \mid class(g) = class(B), g \in A\}|$, $fp = |A| - tp$, $fn = |\{g \mid class(g) = class(B), g \in G \setminus A\}$.

The F1 measure is a harmonic mean of precision and recall, i.e., it takes into account how large is the ratio of true positives it describes and how many instances of the true class it covers. Since the support of the concept may affect the value of F1 measure, we consider F1 measure within 10 frequency bins: (0, 0.1], (0.1, 0.2], ..., (0.9, 1.0]. The average values for some datasets are given in Fig. 3.15.

In "auto" dataset, for example, the most frequent closed itemsets ($fr. \in (0.9, 1.0]$) are located only at the 1st level and have the largest average F1 value. As the frequency decreases the average F1 measure decreases as well. The smallest average F1 corresponds to the itemsets of frequency $(0, 0.1]$.

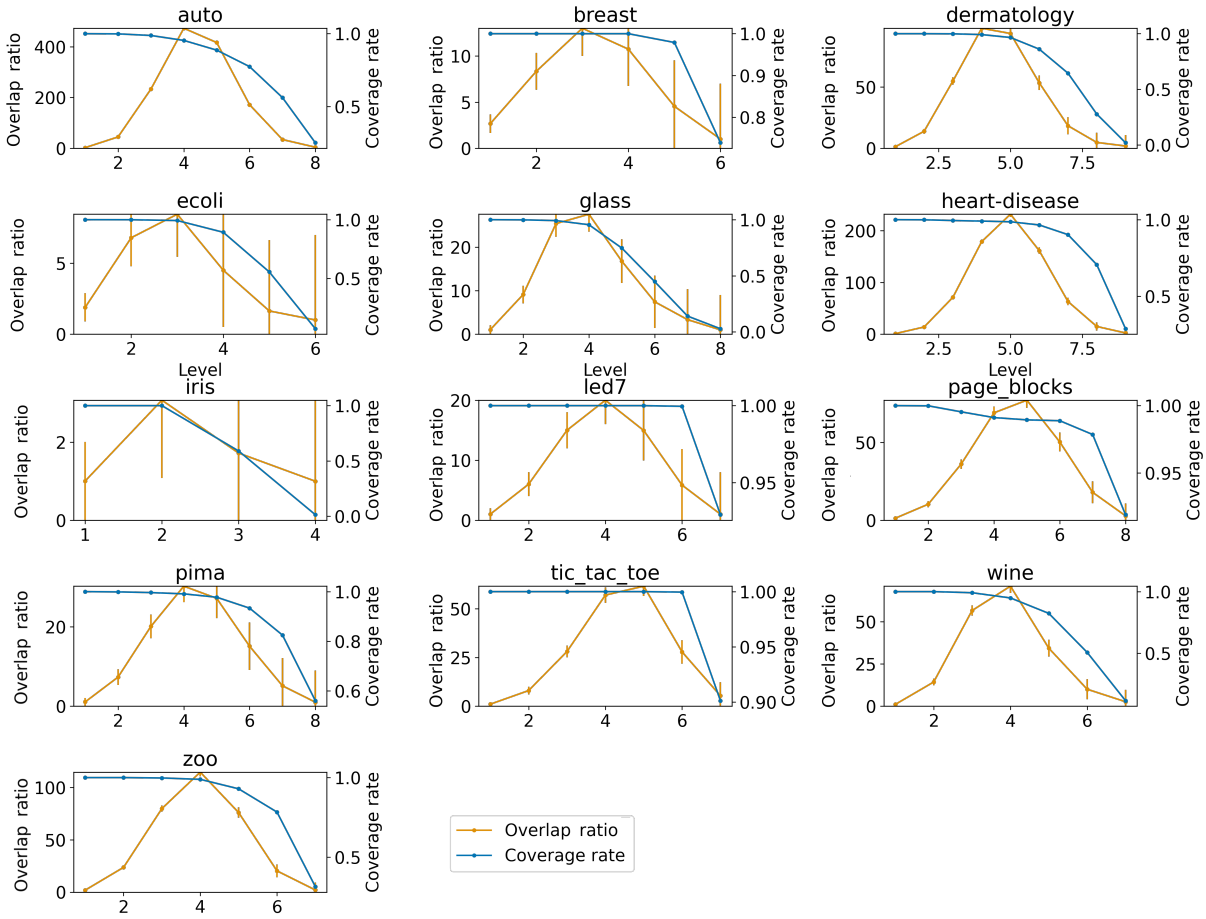


Figure 3.14: Coverage rate and mean overlap ratio of the cells by levels. The vertical lines show the standard deviation of the overlap ratio over cells within a given level

This dependence of the average F1 values on frequency bins of itemsets remains the same over the levels. For example, at level 7 we observe the highest values for itemsets with frequency in the interval $(0.2, 0.3]$ (the largest frequency at the level). Almost for all datasets the values of the average F1 measure remain quite similar for itemsets within a frequency bin over all closure levels.

Thus, the quality of itemsets evaluated w.r.t. F1 measure does not change over the levels and depends rather on the frequency of itemsets.

3.6.6 Case study

In the previous sections we discussed the closure structure in the context of the itemset mining. In this section we demonstrate how the closure structure may be used to obtain exact association rules (implications) in market basket analysis. We consider a dataset from Kaggle repository[6]. It consists of 7835 objects and 167 attributes, the dataset density is 0.026. Frequencies of attributes are shown in Fig. 3.16 in ascending order. This frequency distribution is typical for such kind of data – there are a few very frequent items, e.g., 'whole milk', 'other vegetables', 'soda', 'rolls/buns', 'yogurt', 'bottled water', 'root vegetables', 'shopping bags' and 'tropical fruit', and a large amount of very infrequent items.

The standard approach for computing association rules consists of two stages: (i) enumerating all frequent (closed) itemsets and (ii) computing the rules by splitting each itemset into an antecedent and

[6]https://www.kaggle.com/apmonisha08/market-basket-analysis

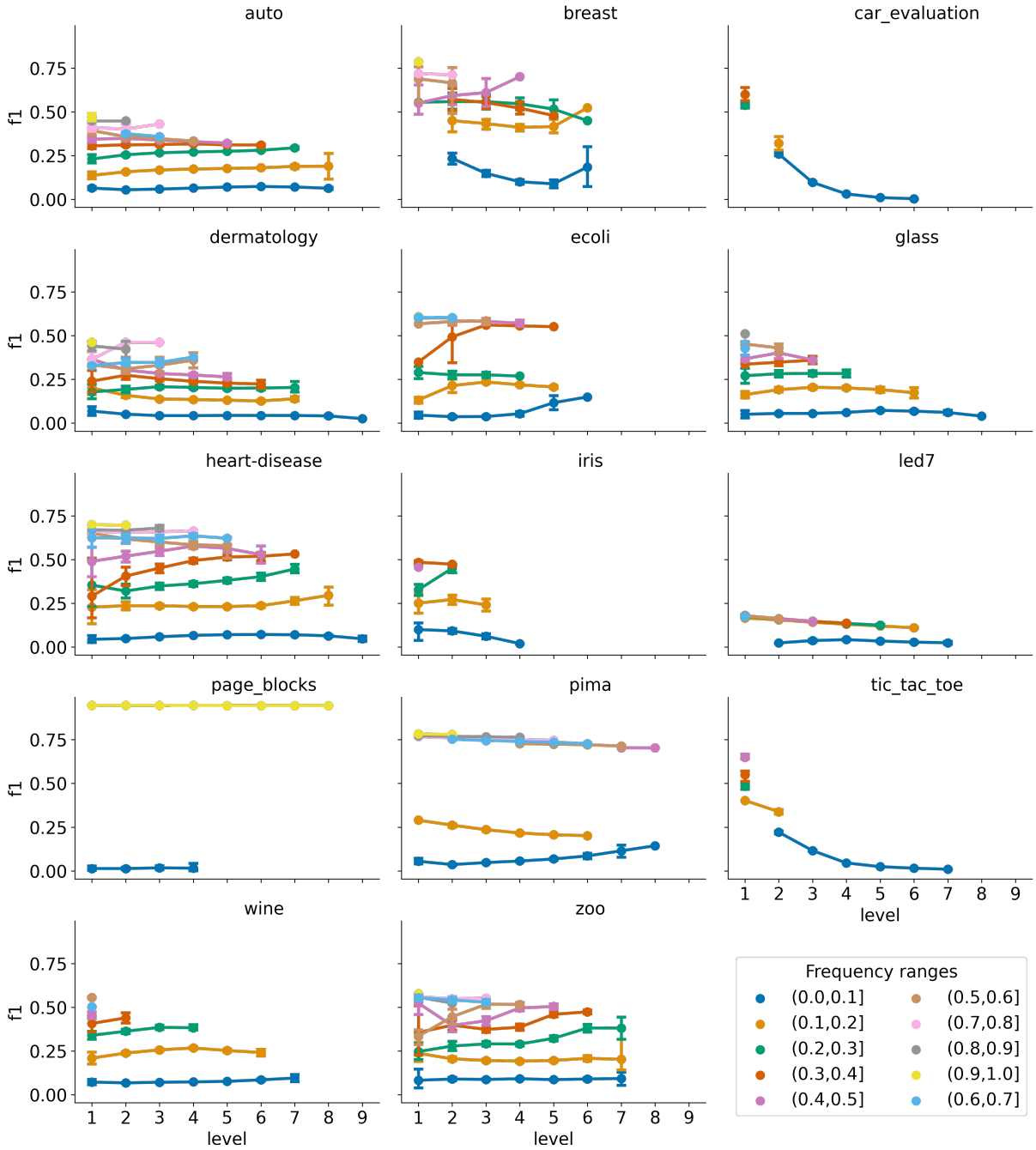


Figure 3.15: Average F1 measure within frequency ranges

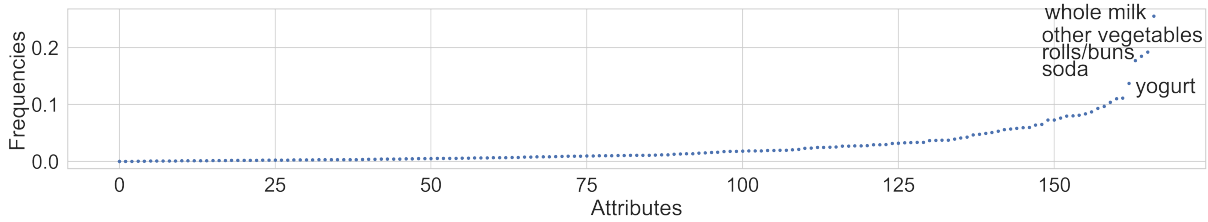


Figure 3.16: Frequency of attributes in ascending order

consequent in all possible ways. We use MLxtend library[7] to generate association rules. The number of enumerated frequent itemsets is 637606, frequency threshold is 0.0005.

To compare the standard approach with one based on the closure structure, we consider rules with confidence equal to 1 (i.e., implications) and order them by lift. The top 5 rules are given in Table 3.4, they are very similar, e.g., the 2nd and 4th rules have the same antecedent and almost the same consequents.

Table 3.4: Top-5 (by lift) association rules with confidence equal to 1 ordered by lift

antecedents	**consequents**	**frequency**	**conf**	**lift**
'frankfurter', 'frozen fish', 'whole milk', 'pip fruit'	'frozen meals', 'other vegetables', 'yogurt', 'tropical fruit'	0.0005	1	1567.0
'grapes', 'frozen fish'	'frozen meals', 'pip fruit', 'other vegetables','tropical fruit', 'whole milk'	0.0005	1	1119.3
'frozen meals', 'other vegetables', 'yogurt', 'frankfurter'	'tropical fruit', 'frozen fish', 'whole milk', 'pip fruit'	0.0005	1	1119.3
'grapes', 'frozen fish'	'frozen meals', 'other vegetables', 'tropical fruit', 'pip fruit'	0.0005	1	870.6
'grapes', 'whole milk', 'frozen fish'	'frozen meals', 'other vegetables', 'tropical fruit', 'pip fruit'	0.0005	1	870.6

Because of the redundancy problem, it is hard to analyze the obtained rules and to get an overall idea about them. However, the closure structure provides an exhaustive set of non-redundant rules and provides a clear representation of them.

Table 3.6 contains the description of the computed closure structure. Each row corresponds to a closure level, each column correspond to a size of closed itemsets. All the implications $X \to Y$ summarized in the k-th row of the table have the smallest possible antecedent (which is a passkey) among all the equivalent implications (i.e., the implications $X^* \to Y^*$ with confidence 1, such that $X^* \cup Y^* = X \cup Y$). The number of implications is computed up to equivalence classes, i.e., if there are several implications with minimum antecedents of size k in an equivalence class, we count only.

A cell contains the number (and the average support) of itemsets of a certain size belonging to a certain level. For example, the 1st level contains 163 itemsets of size 1 (singletons) with the average support 211.4±304.1 and 1 itemset of size 5 having support 2. The 2nd closure level contains itemsets of the sizes from 2 to 14. The itemsets that are located in the diagonal (in gray) can be used to build implications since their consequents are empty. Analyzing the itemsets of size at least 3 of the 2nd level we may conclude that there exists 629 implication with the consequents of size 1 and average support 2.8±1.1, 264 implications with the consequents of size 2 and average support 2.3±0.6, etc. The largest consequents of the implications with antecedent 2 have size 10, 11, 12, their support is 2.

In Table 3.5 we list some implications obtained from itemsets from the 2nd level. Among the obtained implications, we can observe quite specific implications, e.g., if one buys 'specialty chocolate' and 'semi-finished bread' one buys 'pastry' as well, or if one buys 'waffles' and 'nuts/prunes' one buys 'newspapers' as well.

[7] https://pypi.org/project/mlxtend/

Table 3.5: The most frequent implications obtained from the itemsets of the 2nd level

antecedents	**consequents**	**support**
'herbs', 'long life bakery product'	'whole milk', 'root vegetables'	5
'specialty chocolate', 'semi-finished bread'	'pastry', 'whole milk'	5
'citrus fruit', 'rubbing alcohol',	'root vegetables', 'whole milk'	4
'root vegetables', 'rubbing alcohol'	'citrus fruit', 'whole milk'	4
'curd cheese', 'hygiene articles'	'other vegetables', 'yogurt'	4
'grapes', 'frozen fish'	'frozen meals', 'whole milk', 'other vegetables', 'pip fruit', 'tropical fruit'	4
'frozen fish', 'soft cheese'	'yogurt', 'root vegetables'	4
'turkey', 'hamburger meat'	'yogurt', 'whole milk'	4
'spread cheese', 'hygiene articles'	'other vegetables', 'yogurt'	4
'sugar', 'meat spreads'	'yogurt', 'whole milk'	4
'waffles', 'nuts/prunes'	'newspapers', 'whole milk'	4
'specialty fat', 'pip fruit'	'fruit/vegetable juice', 'whole milk'	4
'rice', 'roll products'	'other vegetables', 'fruit/vegetable juice'	4

One of the benefits of the proposed approach is that (i) we deal with easy interpretable association rules, i.e., those that have the smallest possible antecedent and the largest possible consequent, (ii) the closure structure gives the exact number of rules with a chosen sizes of antecedents and consequents, more over these rules are non-redundant.

The proposed approach provides implications that can be more easily analyzed by experts. However, the proposed approach has some drawbacks: the implications are more suitable for "deterministic" data (see Section 3.2), where the closure structure may serve a good tool to obtain the data summary. For noisy datasets or datasets where the appearance of an item in a transaction (row) is rather probabilistic (e.g., an item appears in a shopping cart with a certain probability) computing the exact structure maybe be inappropriate because of its sensitivity to noise (a lot of noisy unstable itemsets may appear). Thus, an important direction of future work is to generalize the notion of closure structure for approximate tiles (biclusters), i.e., dense rectangles in data, to deal with "approximate closures".

3.7 Discussion and conclusion

In this chapter we introduced the "closure structure" of a dataset, with computational aspects, theoretical and practical properties. This closure structure provides a view of the "complexity" of the dataset and may be used for guiding the mining of the dataset. We showed theoretical results about the ordinal nature of the closure structure, and we stated and proved complexity results about the computation of passkeys. Moreover, the GDPM algorithm and its variations are able to compute the closure structure. The application of GDPM over a collection of datasets showed a rather efficient behavior of the algorithm. For synthesizing, the closure structure of a dataset brings the following benefits:

- The closure structure is determined by the closed itemsets and their equivalence classes. It is based on a partition related to the smallest keys — smallest itemsets in an equivalence class — called passkeys and that can represent any closed subset of attributes and related objects without loss.

- The closure structure determines a "topology" of a dataset and is organized around a set of levels depending on the size of the passkeys. This provides a distribution of the itemsets in terms of their associated passkeys. Moreover, the number of levels, called the closure index, can be used to measure the potential size of the pattern space, i.e., given by the minimal Boolean lattice included in the pattern space.

Table 3.6: Summary of the levels of the closure structure (for itemsets with support at least 1)

size of closed itemsets		closure levels 1	2	3	4	5	6	7
1	count	163						
	sup.	163						
2	count		5637					
	sup.		18.3+31.6					
3	count		629	27051				
	sup.		2.8+1.1	6.8+7.7				
4	count		264	7655	28841			
	sup.		2.3+0.6	2.8+1.1	4.6+3.3			
5	count	1	136	3409	10115	7681		
	sup.	2.0+0.0	2.1+0.3	2.4+0.7	2.9+1.1	4.0+2.1		
6	count		71	1483	3526	2258	504	
	sup.		2.1+0.4	2.2+0.5	2.5+0.8	3.0+1.2	3.7+1.6	
7	count		26	677	1211	463	106	3
	sup.		2.1+0.4	2.1+0.4	2.3+0.6	2.6+0.8	3.1+0.9	4.3+0.5
8	count		17	294	376	96	7	
	sup.		2.1+0.2	2.1+0.3	2.2+0.4	2.3+0.6	2.7+0.7	
9	count		14	108	84	14		
	sup.		2.0+0.0	2.1+0.2	2.1+0.3	2.2+0.4		
10	count		3	57	24			
	sup.		2.0+0.0	2.1+0.3	2.1+0.3			
11	count		2	21	6			
	sup.		2.0+0.0	2.0+0.0	2.0+0.0			
12	count		1	8	1			
	sup.		2.0+0.0	2.0+0.0	2.0+0.0			
13	count		1	3				
	sup.		2.0+0.0	2.0+0.0				
14	count		1	3				
	sup.		2.0+0.0	2.0+0.0				
15	count			1				
	sup.			2.0+0.0				
16	count			1				
	sup.			2.0+0.0				

- Given the closure structure and the closure levels, we may understand how itemsets are distributed by levels and know when to stop the search for interesting itemsets with a minimal loss. For example, we discovered that the first levels of the closure structure have better characteristics than the other levels. In particular, the passkeys of the first levels provide an almost complete covering of the whole dataset under study, meaning also that the first levels are the most interesting and the less redundant in terms of pattern mining. In addition, the first levels of the closure structure allow to derive the implications with the smallest antecedents and the maximal consequents, which are considered as the best rules to be discovered in a dataset.
- Finally, we also studied how to use the closure structure with large or very large datasets using sampling. We showed that the closure structure is stable under the sampling of objects, provided that the set of attributes remains identical. Then, a combination of feature selection and object sampling would be a possible way of dealing with very large datasets.

However, the proposed closure structure has some limitations that we aim to improve in future work: the closure structure provides a very rich framework for understanding the content of a dataset and thus

there are many directions for future work. We sketch some of these future investigations below.

- Considering complex datasets such as numerical datasets, how to generalize the closure structure to the mining of numerical datasets of linked data for example? One possible direction to deal with such complex data is to use the formalism of pattern structures, which is an extension of FCA allowing to directly work on complex data such as numbers, sequences, trees, and graphs [KKND11].
- The closure structure is introduced and studied thanks to frequency which depends on the support of itemsets. Along with frequency, it could be very interesting to reuse other measures that would bring different and complementary views on the data, such as e.g., stability, lift, and MDL.
- The closure structure is intended in this paper in terms of pattern mining, and more precisely in terms of itemset mining. The question of the generalization can be raised, for example how the closure structure could be reused in terms of investigations related to pattern mining, such as subgroup discovery, anytime approaches in pattern mining, and also, as already mentioned above, MDL-based pattern mining?
- Many questions remain when we consider the practical point of view. Among these questions, how to define a possibly anytime strategy for pattern mining terms based on the closure structure, knowing in particular that the first levels are the most interesting. Moreover, this strategy could also guide the discovery of the most interesting implications and association rules, and take into account the relations with functional dependencies.
- The GDPM algorithm should be improved in terms of computational efficiency — some of them are already present in Talky-G and DefMe, for example — and as well adapting the algorithm to other datatypes such as numbers, RDF data, as already mentioned above. Finally, another important direction of investigation is to study the possibility of designing a depth-first version of GDPM with an adapted canonicity test, which may improve a lot the computation performance.

4

Pattern mining in binary data

Contents

4.1 Introduction **54**

4.1.1 Pattern types 54

4.1.2 Exploring the pattern search space 55

4.1.3 Interestingness measures 56

4.2 Minimum description length principle in itemset mining **57**

4.3 Greedy strategy to reduce pattern search space **63**

4.3.1 Likely-occurring itemsets 63

4.3.2 Likely-occurring itemsets and related notions 66

4.3.3 Experiments 67

4.4 Adapting the best practices of supervised learning to itemset mining . . . **73**

4.4.1 KeepItSimple: an algorithm for discovering useful pattern sets 73

4.4.2 Experiments 78

4.5 Discussion and conclusion **79**

In the previous chapter, we introduced the closure structure — a theoretical model for describing the intrinsic structure underlying the data. This structure, in a certain sense, is a complete lossless representation of the dataset, since it is based on the complete set of closed itemsets — elements that allow for reconstructing the whole datasets and contain descriptions of each group of objects sharing the same attributes.

The closure structure has important theoretical properties and provides a lot of useful information about the intrinsic data structure. However, the whole structure does not provide a succinct description of the knowledge contained in the dataset. To discover interesting information/knowledge one usually applies a pattern mining method. The closure structure in this case may serve as a supplementary tool to select the most appropriate interestingness measure or constraints to perform pattern mining. Put differently, it helps to understand data and then to decide what might be interesting to search. In this chapter, we focus on problems of pattern mining. We discuss in detail (i) the problems of exponential explosion and (ii) the problem of the alignment of an interestingness measure with the intuitive expectations about the quality of pattern sets. Regarding the first problem, we introduce the notion of so-called likely-occurring itemsets and propose an algorithm for computing them greedily. In the experiments, we show that the proposed algorithm allows for mining high-quality association rules. Regarding the second problem, we propose the KeepItSimple algorithm that uses only a polynomial number of itemsets to build a multi-model description of data relying on a "non-minimum" description length. In the experiments, we show that the proposed modification allows for computing a small set of non-redundant interesting patterns.

4.1 Introduction

A generic objective of pattern set mining is to discover a small set of non-redundant and interesting patterns that describe together a large portion of data and that can be easily interpreted [AH14]. Pattern mining can be summarized into two steps: (i) exploring the pattern search space and (ii) selecting interesting patterns among the discovered ones. Both of these steps is accompanied by a range of issues. Let us discuss them in detail.

4.1.1 Pattern types

To explore the pattern search space, one must first specify the type of patterns. Informally speaking, patterns can be defined as repetitive fragments in data. In binary data one usually deals with either *itemsets* (sets of attributes) or *tiles* (sets of attributes and their extensions). Efficient enumeration of patterns usually implies curbing the pattern explosion.

Frequency is known to be the most popular and widely used constraint to restrict a pattern search space, in particular, an itemset search space. The prevalence of frequency is propelled from its anti-monotonicity [AS94, MTV94], i.e., frequency can be used in enumeration algorithms with the guarantees that all itemsets having frequency above a given threshold will be enumerated (a priori principle). Moreover, frequent itemsets have a very intuitive interpretation, namely "the most interesting itemsets are those that frequently appear in data". However, not all frequent itemsets are interesting, and not all interesting itemsets are frequent. Over the last decades, a wide variety of methods and approaches for enumeration of frequent and frequent closed itemsets has been proposed. Among frequent itemset enumerators, the most well-known are bread-first Apriori [AS94]; FP-growth [HPY00], which builds a suffix tree of frequent patterns and exploits the divide-and-conquer principle; Eclat [Zak00b], which is, as the previous one, a depth-first search algorithm, but instead of horizontal data representation it uses a vertical data representation.

However, frequency-based enumeration of patterns has some limitations, foremost among them are the following: (i) enumerated patterns may be very similar, and (ii) reducing the threshold may cause pattern explosion. The latter means that some infrequent patterns either may remain undiscovered or could be discovered requiring exponential space and time.

Maximal frequent itemsets is a type of patterns that was proposed to represent concisely a set of frequent itemsets. An itemset is *maximal frequent* if it has no frequent supersets. To compute them, a range of algorithms has been proposed, e.g., MaxMiner [BJ98], DepthProject [AAP00], Mafia [BCG01], and GenMax [GZ05].

In [SK01, WK04] it was proposed to use so-called length-decreasing support constraints, where itemset support decreases as a function of their length. This approach allows for reducing the number of low-frequency itemsets.

Closed itemsets — or intents of formal concepts — is another type of itemsets that is widely used to tackle the aforementioned issues. In Section 2.1 and 3.4, we saw that closed itemsets encapsulate up to an exponential number of other itemsets that are equivalent under the closure. Thus, the restriction of itemsets to closed ones allows for reducing drastically the pattern search space without any additional user-specified constraints.

However, even for closed itemsets (formal concepts), the pattern explosion problem remains acute. To mitigate this problem, the number of patterns is usually further reduced by imposing additional constraints.

Frequency is a commonly used measure to restrict the number of enumerated closed itemsets. Many algorithms for enumeration of closed itemsets adopt the ideas of the frequent itemset enumerators, the most famous algorithms are Close [PBTL99c], Charm [ZH02], and Closet [PHM00]. Later, very efficient algorithms, called LCM and Alpine, for mining closed itemsets have been proposed. The LCM [UKA05] algorithm, as the majority of the algorithms for enumeration of frequent itemsets, requires the frequency threshold to be set in advance. Its cousin Alpine [HI16] is devoid of this drawback. This algorithm is anytime and computes itemset of decreasing frequency. Being halted at any time, it returns a complete set of frequent closed itemsets for a certain frequency threshold. We already discussed these

algorithms earlier in Section 3.2. A detailed overview of them can be found in [AH14]. A comparative study of the algorithms for enumeration of closed itemsets in the FCA framework can be found in [KO02].

In the scientific literature, there exists a lot of other types of itemsets that can reduce the size of the itemset search space as well, e.g., non-derivable itemsets [CG02], i.e., itemsets whose frequency cannot be derived using inclusion/exclusion rules; margin-closed [MTU11], i.e., itemsets that do not have a superset with the frequency that differs by less than α; free itemsets [BBR03], i.e., minimal itemsets by inclusion among those that have the same extension, and others.

A *tile* [GGM04] is another type of patterns that, along with itemsets, is widely used in pattern mining for binary data. Informally speaking, tiles are rectangles in data containing 1's. The ratio of 1's in the tile is called tile *density*. Together with frequency and area, density is widely used as a constraint for enumerating tiles.

Formal concepts can be considered as the maximal exact tiles, i.e., the tiles that are fully filled with 1's and cannot be extended with rows or columns also containing only 1's without lowering their density. In a more general view, tiles can be considered as rectangles not only in binary but also in numerical data. Tiles are widely used in knowledge discovery and data mining, and appear in the literature under different names (and have different properties), e.g., co-clusters [DMM03], biclusters [Mir96, MO04], Boolean factors [MV11].

It is noteworthy that a lot of types of bicluster can be formalized in the framework of either FCA or its generalization called the pattern structures, they were introduced in Section 2.1 and 2.2, respectively. Biclusters in binary data are closely related to FCA [PRB05, MPM+12]. Moreover, FCA provides powerful tools for generalization of biclusters to triadic clusters [MK11], and to clusters in numerical data [KKN11a, JCN19, CN14].

In linear algebra, there exists an analog of tiles (or biclusters) called *Boolean factors*. Boolean Matrix Factorization (BMF) [MV11] offers a range of algorithms that are aimed to find an approximate decomposition of a binary matrix of size $N \times M$ into the Boolean product of two matrices of a lower dimension K, i.e., into matrices of sizes $N \times K$ and $K \times M$, the columns of the first matrix and the rows of the second one describe which rows and columns constitute each of K tiles.

Recently, an approach for mining arbitrarily shaped geometrical patterns has been proposed [FvL20]. This approach is applicable to the datasets where the order of columns and rows is important, e.g., images.

4.1.2 Exploring the pattern search space

Usually, the pattern search space, no matter what type of patterns is chosen, contains exponentially many patterns, but only a few of them are useful and interesting for analysts. Thus, during the first stage of pattern mining — pattern search subspace discovery — it is crucially important to discover as few uninteresting patterns as possible. Moreover, efficiency of the second stage — selection of interesting patterns among the discovered ones — is often determined by the quality of the algorithms applied at the first stage. Intuitively, the best algorithms for pattern search space discovery are those that are tailored for a specific interestingness measure. With this regard, we divide these algorithms into two groups: (i) those that discover a pattern search subspace using a specific objective (interestingness measure), and (ii) general-purpose algorithms, i.e., that are not tailored to a specific interestingness measure and allow for applying a wide range of interestingness measures at the second stage.

The methods of the first group are more efficient in the sense that they generate only those patterns that comply with a chosen interestingness measure and generate a small number of uninteresting patterns w.r.t. this measure.

The main representatives of this group are *sampling* algorithms. Instead of enumerating patterns that satisfy certain constraints, these methods sample patterns according to a given quality measure. A lot of pattern samplers belongs to the Markov Chain Monte Carlo methods [BG09, BGG10], thus may converge slowly. Some approaches [BLPG11, BMG12] do not require a long simulation phase and mine patterns directly from data. However, these approaches are limited in the quality measures (constraints) that they support. In [DvLDR17] the authors consider sampling as a constraint satisfaction problem and propose a sampling approach that supports a much wider variety of quality measures (constraints) according to which itemsets are sampled.

Sampling methods provide only an approximate solution. Another way to discover patterns complying with a chosen measure is to enumerate patterns only in those subspaces of the pattern search space that contain potentially interesting patterns. We will discuss in detail one of representatives of this group, the SLIM [SV12] algorithm, in Section 4.2.

In the cases where a specific algorithm for a chosen interestingness measure does not exist, the only resort is general-purpose enumeration algorithms. Frequent closed itemsets are known to be one of the most popular type of patterns enumerated in such cases. However, the problem of itemset explosion remains acute for the frequency-based itemset enumeration. In Section 4.3 we propose an alternative algorithm for enumeration of itemsets that allows for reducing the size of the discovered itemset search subspace.

4.1.3 Interestingness measures

The pattern search space discovery, discussed above, constitutes the first stage of pattern mining. The aim is to enumerate candidates to be evaluated in the second step and, as much as possible, focusing on the promising candidates rather than those that will be discarded in the second step. The aim of the second step is to select the most interesting patterns w.r.t. a chosen interestingness measure.

Interestingness is an inherently subjective notion, and in pattern mining there exists a wide variety of approaches to its formalization. The measures for assessing pattern interestingness can be divided into two groups: (i) measures for *patterns* (applied to each pattern individually), and (ii) measures for *pattern sets* (applied to the whole pattern set). Itemset mining is the area of pattern mining with the most diverse approaches to defining interestingness. Thus, we discuss the specific approaches in the context of itemset mining.

Interestingness measures for patterns. A very straightforward method to assess interestingness is to apply an interestingness measure to evaluate each pattern individually. In [GH06] the authors propose to categorize interestingness measures into three groups: *objective*, *subjective* and *semantics-based*.

Objective measures are based on datasets and do not require any background knowledge or user feedback. Often these measures rely on statistics, probability-based estimates, or some measures from the information theory. To apply the measures based on probability or statistics, one makes an assumption about the distribution underlying the data.

The most common measures that rely on assumptions are based on the independence model and the maximum entropy principle.

Assuming that all attributes are independent, one may search for patterns whose observed frequency deviates a lot from one obtained under the independence model [BMS97, AY98]. The assumption on the independence of attributes may be too naive, and this model can be generalized to the itemset independence model [MW99].

The maximum entropy (MaxEnt) principle is considered to be more sound (less naive) from the theoretical perspective. The MaxEnt distribution has the maximal uncertainty among all the distributions that meet the imposed constraints [Jay82], i.e., by using the MaxEnt distribution we do not incorporate into the model what we do not know. Then, as with the independence model, one may compare the observed itemset frequency with one obtained with the MaxEnt distribution [PMS03].

The expected probability of itemsets can be also computed using graph-based models. In [JS04] a Bayesian network (a directed acyclic graph) is used to represent dependencies between items and then to select itemsets whose estimated and observed frequencies deviate a lot. Later, a less restrictive approach based on Markov Random Field [WP06] (an undirected graph that exploits conditional independence relations among the items) was proposed to summarize frequent itemsets. The significance of itemsets can be also evaluated using swap-randomization techniques [GMMT07].

Thus, these measures rely on particular assumptions and, in a certain sense, may be considered subjective.

Subjective measures take into account user knowledge/expectation/assumptions or user feedback. This information can be integrated in the beginning or collected and adjusted during mining, as new patterns are discovered.

Apart from user-specific background knowledge, one may use domain knowledge, i.e., the knowledge that is not related to a particular user. The approaches that use domain knowledge make the third group of interestingness measures, called semantic-based [GH06]. One of the common representatives of this group is utility-based measures [HM07]. For example, in market basket analysis, as domain knowledge one may use the prices of items, then, as an interestingness measure, a function maximizing the purchase value can be chosen.

The pattern mining approaches based on interestingness measures for individual patterns have a very important drawback: they tend to assign almost the same values to almost the same patterns. As the result, when evaluating patterns individually and selecting the best, the set of selected patterns is redundant, i.e., many patterns are similar to each other.

To tackle this issue it was proposed to assess pattern interestingness by taking into account other patterns, i.e., instead of evaluating each pattern independently, to evaluate a pattern set as a whole.

Interestingness measures for pattern sets. In contrast to the considered above interestingness measures, where each pattern is assessed individually, approaches for pattern sets evaluate patterns w.r.t. other patterns. Put differently, instead of pattern interestingness, they assess the interestingness of a pattern set. The pattern set is mined iteratively, in a greedy manner, the estimates of pattern interestingness can be adjusted each time a new pattern is added to the set.

The most common approaches for pattern set mining are based on maximum entropy [MVT12, DB11] and minimum description length (MDL) [VVLS11, SV12, SK11] principles. Both of them measure interestingness relying on information theory [CT12]. As it was mentioned in the previous section, the maximum entropy distribution possesses the largest uncertainty among all the distributions consistent with the given constraints [Jay82], i.e., it does not contain any additional information that might be wrong.

The methods based on MDL [Grü07] is one of the actively developing directions of pattern mining. Recently, the MDL-based approaches have been proposed to mine a large variety of pattern types, e.g., graphs [SV19, BCF20], sequences [LKPC14], itemsets [VVLS11], numeric hyper-rectangles [WDKG14], tiles [TV12a], images [FvL20], etc.

In this section we focus on MDL-based methods for itemset mining and consider the seminal work in the MDL-based itemset mining.

4.2 Minimum description length principle in itemset mining

In the context of MDL, interesting itemsets are those that compress data well. Usually, in itemset mining, one uses the crude (two-part) version of MDL. One of the most common two-part description lengths for mining itemsets was proposed in [VVLS11] and now is adapted in other approaches to the MDL-based itemset mining and related tasks [ATVF12, SK11, SV12]. As a model, one uses a code table, containing itemsets and the associated code words.

More formally, the total description length is given by

$$L(D, CT) = L(CT) + L(D|CT),$$

where CT is a two-column code table, whose left- and right-hand columns contain itemsets over attribute set M and the associated code words, respectively. One distinguishes a special type of code tables called *standard*. In the standard code table, all patterns are singletons. To encode a dataset using itemsets from a code table, one should consider these itemsets one by one from the top to the bottom of the code table and replace each occurrence of the current itemset with the corresponding code word. The items that are replaced with an itemset are said to be *covered*, and they cannot be covered by other patters, i.e., each item in a dataset can be covered by no more than one itemset. The fragments of data encoded by singletons are considered as uncovered.

Roughly speaking, the basic idea of encoding with the code tables is to replace the uniform codes with the codes of variable lengths. The advantage of using the variable-length codes rather than uniform is that more frequent codes in data will have shorter code words. However, one should keep the information

about the fragment that corresponds to each code word. A code table is a dictionary that contains the encoding fragment and the associated codes of variable lengths. Thus, the goal is to find which fragment should be encoded and to select the code words of the lengths ensuring the shortest total description length.

Example. In Fig. 4.1 (upper row) we show a dataset where each attribute is encoded with the uniform codes. Instead of showing the code words themselves, we represent them as rectangles and focus on the rectangle widths. The widths are proportional to the lengths of the corresponding code words.

Then, in the second row, we show how the length of a data can be reduced when the code words of the fixed length are replaced with the code words of variable lengths. In MDL, the code table containing only singletons as patterns is called *the standard code table*. The lengths of the code words are inversely proportional to their usage in data, e.g., the most frequent items a and b have the shortest code words in the standard code table, while items d, e and f have the longest code words, since they are used only twice each to encode the data.

Comparing the datasets encoded with the uniform codes and with the variable-length codes from the standard code table, we can see that chosen properly variable lengths allow for reducing the description length of data (i.e., the sum of the widths of the rectangles). However, it is important to notice that the uniform codes are not used in the standard code tables ST. The dictionary that associates the uniform codes with the variable-length codes (the third column) is not taken into account in the total description length $L(D, CT)$. It means that, formally, the original data cannot be fully decoded using only the standard code table.

The standard code table, in a certain sense, may appear useless, since it associates the code words of singletons to themselves. However, it is only the starting point of itemset mining. The code tables in general contain itemsets, where the left-hand column contains the description of itemset encoded with the codes of singletons from the standard code table, and the right-hand column contains the associated code words. In the last row, we give an example of such a code table and the encoding of data with these code table.

As it can be seen from the figure, the length of the dataset is reduced drastically, while the length of the code table, containing the itemsets and the associated code words, is larger than the standard one. As in the case of the standard code table, we do not take into account the associations between the variable-length code words and the corresponding singletons (we show it in the "implicit code table"). Instead, the code words in the code tables located in the right-hand column are described by the code words associated with the singletons and having lengths inversely proportional to their frequency. The latter simplifies the encoding but, formally, does not allow to decode the dataset without loss of information. This problem can be easily solved by adding a term corresponding to this "implicit code table" to $L(D, CT)$. However, it is not the case for the commonly used models, e.g., ones used in Krimp and Slim.

Let us proceed to the question of finding an optimal code table.

The problem of finding the MDL-optimal pattern set is known as *minimal coding set problem* [SV12] and is formulated as follows: let M be a set of attributes and let D be a dataset over M, Cover is a cover algorithm, and $\mathcal{F} \subseteq \mathcal{P}(M)$ is a collection of candidate patterns. Find the smallest set of patterns $\mathcal{S} \subseteq \mathcal{F}$ such that for the corresponding code table CT the total compressed size, $L(D, CT)$, is minimal.

The number of candidates is bounded from above by $2^{\min(|G|,|M|)}$, the number of possible combinations of these candidates is bounded from above by $2^{2^{\min(|G|,|M|)}}$. Moreover, code word lengths can be assigned in many ways within each pattern set. It is clear that the minimal coding problem is intractable in practice. However, there are some greedy approaches to find an approximate solution.

In practice, the MDL-based itemset mining consists of two phases: (i) computing a candidate pattern set, (ii) selecting greedily patterns providing a shorter total description length. A pattern is considered as MDL-optimal if it allows for better compression (a shorter description length). The cornerstone of an efficient compression is *a covering strategy*, i.e., a compression scheme that defines how patterns will serve to compress data, and *the code length function*, i.e., the principle of associating a code word length to a pattern. The phases (i) and (ii) can be performed independently (Krimp) or in a loop (Slim), both approaches are sketched in Fig. 4.2. Let us consider these two phases in detail.

Phase I. Computing candidates (pattern space exploration). In Krimp, the candidates are all (frequent) closed itemsets. The number of closed itemsets is bounded from above by $O(2^{\min(|G|,|M|)})$, and,

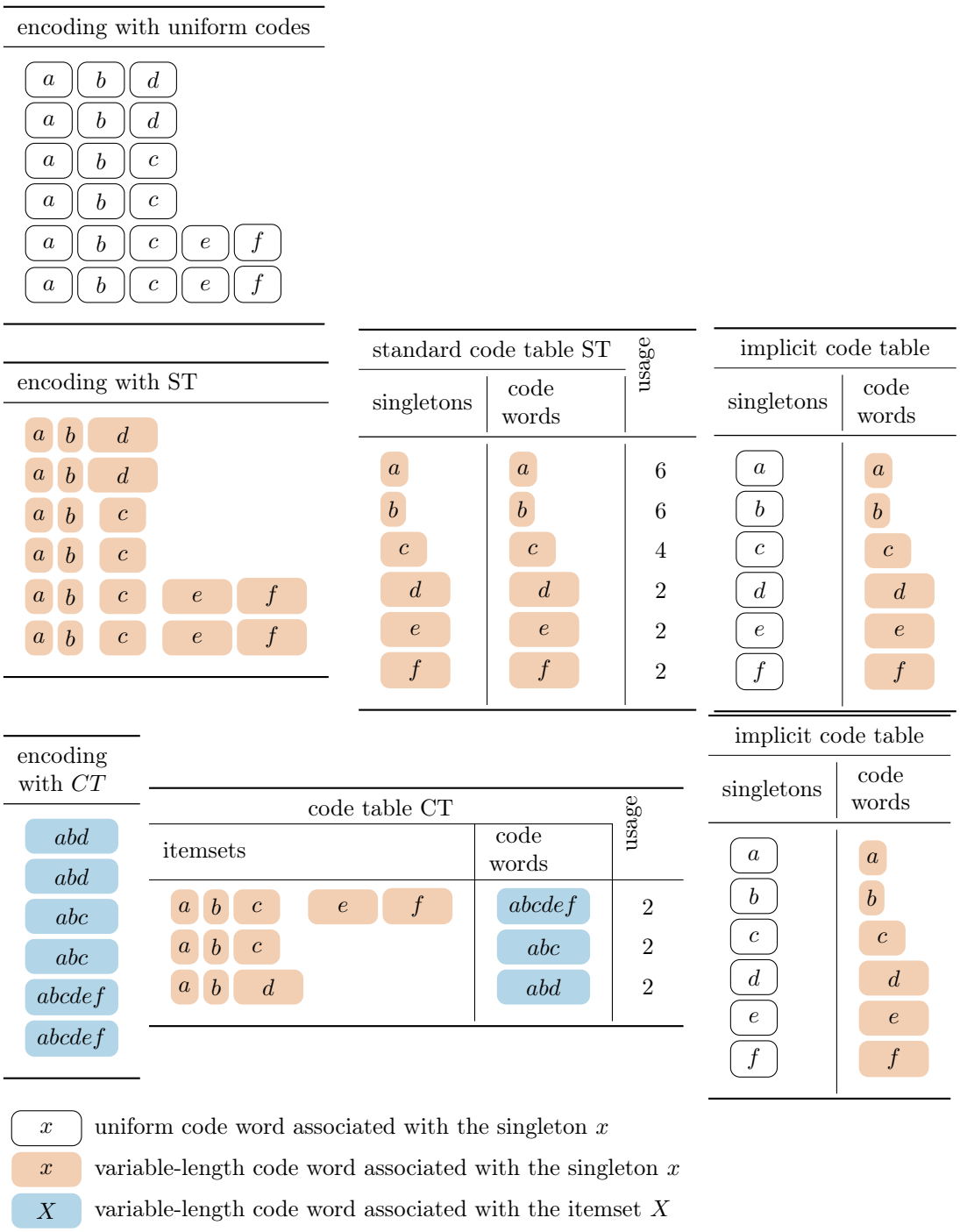


Figure 4.1: Upper row: dataset encoded with uniform codes, middle row: dataset encoded with the standard code table, the standard code table, and the implicit code table, bottom row: dataset encoded with a code table, the code table, and the implicit code table. The two-part description length is the sum of the rectangles corresponding to the encoded data and the rectangles located in the code table

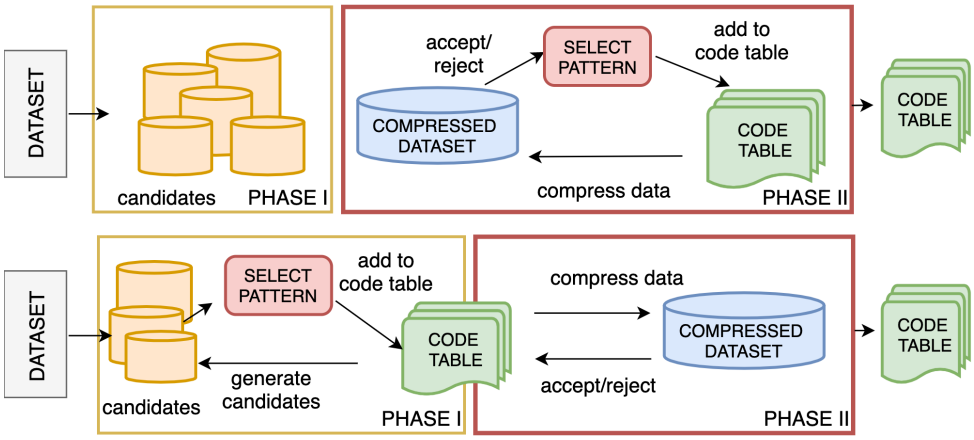


Figure 4.2: The phases of MDL-based itemset mining, where candidates are computed before (top, Krimp) / recomputed in the process (bottom, Slim) of the length minimization, adapted from [sli]

in practice, it contains a lot of quite similar itemsets. To avoid excessive computations and generate only itemsets useful for compression, an iterative approach for candidate computing, called Slim, has been proposed in [SV12]. The idea is that at each iteration, once a new state of the code table is computed, the set of candidates is recomputed. The new set of candidates is composed of the union of all pairs of itemsets from the current code table. We give an example of the candidates used by Krimp and Slim in Fig. 4.3.

Krimp computes all candidates at ones, then filters. The set of all closed itemsets of frequency at least 2 — the candidates of Krimp — contains only 3 itemsets.

Slim computes the candidates gradually, by considering the pairwise union of itemsets in the current code table. Column "Step n" contains the itemsets from the code table at iteration n. Column "Candidates" contains candidates computed by the pairwise union of the itemsets from the current code table. New itemsets, entering to the code table of Slim at iteration n, are highlighted in bold in the column "Step n". Candidates, newly added to the candidate set, are highlighted in bold in the columns "Candidates". The best candidate, that will be considered as a member of the code table at the next iteration, is highlighted with a light-blue background. In the beginning, the candidates are pairs of singletons, i.e., ab, ac, ..., ef. At the end of the first iteration, the pattern ab is added to the code table. At the second step, the set of the pairwise union of patterns ab, c, d, e and f makes a new candidate set, etc. At the last step, the candidates are computed by combining abc, abd and ef. Among the candidates, only $abcef$ occurs in data. Adding it to the code table does not improve the total length, $L(D, CT)$, thus, it is removed from the code table. The algorithm stops.

Despite the fact that the number of candidates in both approaches is theoretically exponential, in practice, Slim generates a much lower number of candidates [SV12]. The Slim candidate generation is more efficient since the candidate exploration is biased towards the subspaces that contain local optima.

Now, let us consider one iteration of Phase II, i.e., the length minimization, when the prospective members of the code table, arranged in a fixed order, are analyzed.

Phase II. The compression principle. Krimp and Slim are based on the same model. Let us consider first this model in detail, and then discuss how this model is used to compress data. Given a set of candidates, one iteration of the minimization is the same for Krimp and Slim. The *standard candidate order* is the arrangement of itemsets by *frequency* $\downarrow$, *length* $\downarrow$ and *lexicographically* $\uparrow$ ($\downarrow/\uparrow$ denotes the descending/ascending order, respectively). The *standard cover order* is the arrangement of itemsets by *length* $\downarrow$, *frequency* $\downarrow$ and *lexicographically* $\uparrow$.

Building the code table consists in the following. Given a set of candidates $\mathcal{C}$ in the standard candidate order, add one by one the candidates to the code table CT. In the code table patterns are arranged in the standard cover order. In the beginning, the dataset encoded with the singletons is considered as uncovered (see example in Fig. 4.1, the second row). Then, to compress D, patterns from CT are used

KRIMP candidates
abd
abc
$abcef$

SLIM candidates

Step 1	Candidates				
	b	c	d	e	f
a	$\boldsymbol{ab}$	$\boldsymbol{ac}$	$\boldsymbol{ad}$	$\boldsymbol{ae}$	$\boldsymbol{af}$
b		$\boldsymbol{bc}$	$\boldsymbol{bd}$	$\boldsymbol{be}$	$\boldsymbol{bf}$
c			$\boldsymbol{cd}$	$\boldsymbol{ce}$	$\boldsymbol{cf}$
d				$\boldsymbol{de}$	$\boldsymbol{df}$
e					$\boldsymbol{ef}$
f					

Step 2	Candidates			
	c	d	e	f
$\boldsymbol{ab}$	$\boldsymbol{abc}$	$\boldsymbol{abd}$	$\boldsymbol{abe}$	$\boldsymbol{abf}$
c		cd	ce	cf
d			de	df
e				ef
f				

Step 3	Candidates		
	ef	c	d
ab	$\boldsymbol{abef}$	abc	abd
$\boldsymbol{ef}$		$\boldsymbol{cef}$	$\boldsymbol{cdef}$
c			cd
d			

Step 4	Candidates	
	ef	d
$\boldsymbol{abc}$	$\boldsymbol{abcdef}$	$\boldsymbol{abcd}$
ab	$abef$	abd
ef		def
d		

Step 5	Cand.
	ef
abc	$abcdef$
$\boldsymbol{abd}$	
ef	

Figure 4.3: The candidates generated by KRIMP (upper), discovered gradually by SLIM (bottom)

to cover uncovered yet fragments of data. The pattern probability under the covering scheme is given by

$$P(X) = \frac{usage(X)}{\sum_{Y \in CT} usage(Y)}. \tag{4.1}$$

where *usage(X)* is the frequency of X in the covering ($usage(X) \leq freq(X)$). For example, in Fig. 4.4 (right), $usage(abc) = 2$ and $freq(abc) = 4$. The code word length of X is estimated using the Shannon code length, i.e., $L(code(X)) = -log(P(X))$, which ensures an optimal compression of the dataset D (short lengths are assigned to more likely occurring itemsets) [Grü07]. For details, see Section 2.3. The code length of itemsets may be considered as itemset importance. The itemsets with shorter lengths are more important since they capture more "general" regularity in data.

Thus, the length of the code table is given as follows:

$$L(CT|D) = \sum_{X \in CT} L(X|ST) + L(code(X)),$$

where pattern code length $L(code(X))$ is proportional to its probability (see Equation 4.1), whereas items from X are encoded by means of the standard code table ST and stored in the left-hand column:

$$L(X|ST) = \sum_{x \in X} \frac{l(x)}{\sum_{x \in M} l(x)} = \sum_{x \in X} \frac{\log(freq(x))}{\sum_{x \in M} \log(freq(x))}. \tag{4.2}$$

The length of the data, encoded with the itemsets from the code table CT is given by $L(D|CT) = \sum_{X \in CT} usage(X) L(code(X))$.

Example. Let us consider the encoding with the code tables computed by KRIMP and SLIM, they are given in Fig. 4.4. KRIMP returns 3 itemsets, namely $abcef$, abc and abd. To encode data they are arranged in the standard cover order. Then, one by one, they are used to cover the dataset: $abcef$ covers entirely objects g_5 and g_6. The second pattern abc is used to cover g_3 and g_4, i.e., the fragments that contain uncovered items a, b, c. The last pattern abd covers uncovered yet objects g_1 and g_2. The usage of all patterns is equal to 2, thus the lengths of the associated code words are the same and equal to $-\log 2/6$. The total length $L(D, CT)$ is composed of two components $L(D|CT)$ and $L(CT)$, which are computed as follows:

KRIMP code table		usage
itemsets	code words	
a b c e f	$abcdef$	2
a b c	abc	2
a b d	abd	2

encoding with the KRIMP code table	
g_1	abd
g_2	abd
g_3	abc
g_4	abc
g_5	$abcdef$
g_6	$abcdef$

SLIM code table		usage
itemsets	code words	
a b c	abc	4
a b d	abd	2
e f	ef	2

encoding with the SLIM code table		
g_1	abd	
g_2	abd	
g_3	abc	
g_4	abc	
g_5	abc	ef
g_6	abc	ef

Figure 4.4: Encoding of data with the code tables computed by KRIMP (top) and SLIM (bottom). Patterns in the code tables are arranged in the *standard cover order*

$$L(D|CT) = 2 \cdot \Big(L(code(abd)) + L(code(abc)) + L(code(abcef))\Big),$$

$$L(CT) = \sum_{X \in \{abcef, abc, abd\}} \underbrace{L(X|ST)}_{\substack{\text{the left-hand}\\ \text{column length}}} + \underbrace{L(code(X))}_{\substack{\text{the right-hand}\\ \text{column length}}}$$

$$= \underbrace{3 \cdot (l(a) + l(b)) + 2 \cdot l(c) + l(e) + l(d) + l(f)}_{\substack{L(abcef|ST)+L(abc|ST)+L(abd|ST)\\ \text{the left-hand column length}}} + \underbrace{L(code(abd)) + L(code(abc)) + L(code(abcef))}_{\text{the right-hand column length}}.$$

Let us consider how to cover the dataset by patterns abc, abd and ef, returned by SLIM. We begin with the top-pattern abc and cover uncovered objects $g_3 - g_6$. Items e and d in g_5 and g_6 remain uncovered. Then abd is used to cover g_1 and g_2. Finally, the last pattern ef is used to cover g_5 and g_6. Note that if, instead of ef, $abcef$ was taken, it would not be used to cover g_5 and g_6, since abc's are already covered. The lengths of the two parts of $L(D, CT)$ are given by

$$L(D|CT) = 2 \cdot \Big(L(code(abd)) + L(code(ef))\Big) + 4 \cdot L(code(abc)),$$

$$L(CT) = \underbrace{2 \cdot (l(a) + l(b)) + l(c) + l(d) + l(e) + l(f)}_{\substack{L(abd|ST)+L(abc|ST)+L(ef|ST)\\ \text{the left-hand column length}}} + \underbrace{L(code(abc)) + L(code(abd)) + L(code(ef))}_{\text{the right-hand column length}}.$$

The described covering procedure is performed each time a new candidate is appeared in the code table.

4.3 Greedy strategy to reduce pattern search space

In the previous sections of this chapter, we discussed the problem of itemset (or pattern) explosion. To make pattern mining applicable in practice, one usually restricts the pattern search space. The latter can be done in several ways. The first common solution is to use a specific algorithm for pattern search space discovery adapted to a chosen interestingness measure. The second solution is to impose monotone or anti-monotone constraints on patterns. In itemset mining, one usually uses frequency as an anti-monotone constraint. However, the set of frequent (closed) patterns is usually redundant. For example, Krimp and Slim — MDL-based itemset miners — minimize the same length but select itemsets from different sets of candidates. The comparative study of these approaches showed that the itemset search space discovery w.r.t. an interestingness measure is more efficient than the itemset search space discovery w.r.t. the frequency-based constraint since it allows for achieving better compression with a moderate number of candidates [SV12].

Nevertheless, considering the itemset mining problem in more general settings, e.g., mining association rules and implications, frequent itemsets is usually the only (universal) remedy for the pattern explosion problem.

In this section, we introduce the notion of *likely-occurring* itemsets and propose a greedy approach to itemset search space discovery that allows for reducing the number of arbitrary or closed itemsets. This method provides itemsets that are useful for different objectives and can be used as an additional constraint to curb the itemset explosion. In experiments we show that the method is useful both for MDL-based itemset mining and for computing good-quality association rules.

4.3.1 Likely-occurring itemsets

To reduce the itemset search space, we propose an additional constraint that consists in considering only the itemsets whose observed probability is greater than the estimated one. The estimations are made under the independence model. We give the details on the chosen independence model below.

Let $\mathbb{K} = (G, M, I)$ be a formal context. The (empirical or observed) probability of an itemset $X \subseteq M$ is given by $P(X) = fr(X) = |X'|/|G|$.

Definition 16. *An itemset X is called* likely-occurring (LO) *if there exists an item (attribute) $m \in X$ such that $P(X) > Q \cdot P(X \setminus \{m\}) \cdot P(\{m\})$, where $Q \geq 1$.*

The parameter Q controls how large the difference between the observed probability $P(X)$ and the estimated probability $P(X \setminus \{m\}) \cdot P(\{m\})$ of the itemset X should be. The least restrictive constraint, i.e., $Q = 1$, requires the observed probability to be greater than the estimated one. The larger values of Q are more restrictive, i.e., requires the observed probability to be much larger than the estimated one.

According to the definition above, at most $|X|$ splittings should be enumerated to check whether an itemset X is LO or not. To make it more tractable in practice, we propose a relaxation of the LO itemset and a greedy approach for its computing, where one needs to check only one splitting per itemset. Let us proceed to this definition.

Definition 17. *Let $M^* = \{m_1, m_2, \cdots, m_k\}$ be a set of attributes arranged in the order of decreasing frequency, i.e., $fr(m_i) \geq fr(m_j)$ for any $i \leq j$. Then, we consider an itemset X as* likely-occurring *if $P(X) > Q \cdot P(X \setminus \{m_{last}\}) \cdot P(\{m_{last}\})$, where $m_{last} \in X$ is the last item, i.e., one of the minimal frequency among all itemsets in X, and $X \setminus \{m_{last}\}$ is likely-occurring.*

The empty itemset $\emptyset$ is considered to be likely-occurring by default.

We propose an algorithm to compute LO itemsets using Definition 17, its pseudocode is given in Algorithm 4. This algorithm computes gradually LO itemsets by considering one by one attributes of decreasing frequency. Apart from the threshold Q on the difference in probabilities, the algorithm also supports threshold F on frequency. By default, we use minimal restrictions, namely $Q = 1$ (we require the observed probability to be greater than the estimated one) and $F = 0$ (we do not impose any frequency constraints).

Algorithm 4 ComputeLO

Procedure Merge (*node*,*candidate*)

Input: *node*, current node
candidate, candidate node

1: $I_n \leftarrow node.itemset$
2: $I_c \leftarrow candidate.itemset$
3: **if** $|I_n \setminus I_c| > 0$ **then**
4: $extent \leftarrow (I_c \cup I_n)'$ {computing shared objects}
5: **if** $extent = I_c'$ **then**
6: $I_n \leftarrow I_n \cup I_c$ {if I_n is included into all objects as I_n, just extend I_c}
7: **else if** $\left(\frac{|extent|}{|G|} > Q \cdot \frac{|I_c'|}{|G|}\frac{|I_n'|}{|G|}\right)$ **and** $|extent| \geq F$ **then**
8: **for all** $child \in node.children$ **do**
9: $merge(child, candidate)$
10: **end for**
11: **if** $candidate \notin \mathcal{T}$ **then**
12: $node.children.add(candidate)$
13: **end if**
14: **end if**
15: **end if**

Input: (G, M, I) formal context
F, frequency threshold
Q, threshold on LO (see Definition 17)
Output: $\mathcal{T}$, a tree composed of LO/LOC-itemsets

1: $\mathcal{T} \leftarrow createEmptyTree()$
2: $root \leftarrow \mathcal{T}.getRoot()$
3: $M^* \leftarrow sortByDescendingFrequency(M)$
4: **for all** $m \in M^*$ **do**
5: $candidate \leftarrow m$ {for LO itemsets}
$candidate \leftarrow m''$ {for LOC itemsets}
6: **for all** $child \in root.children$ **do**
7: $merge(child, candidate)$
8: **if** $candidate \notin \mathcal{T}$ **then**
9: $root.children.add(candidate)$
10: **end if**
11: **end for**
12: **end for**
13: **return** $\mathcal{T}$

Example. Let us consider how the algorithm works using a small dataset from Fig. 4.5. The attributes are already arranged by frequency, in decreasing order. We use default settings, viz. $Q = 1$ and $F = 0$. The execution tree is given in Fig. 4.6. Each node of the tree is labeled by the corresponding itemset. The numbers indicate the order in which itemsets were created, e.g., itemset *ad* was created the 7th, while *ade* was created the 14th.

Already created itemsets can be updated. For example, LO itemsets *bcd* and *cd* (that were created the 8th and 10th) are replaced with *bcde* and *cde* at the 16th and 20th steps, respectively. The itemsets are updated by executing lines 5-6 of the Merge procedure given in Algorithm 4.

However, the set of the generated LO itemsets is redundant, i.e., there is a lot of quite similar itemsets or those that belong to the same equivalence class, e.g., *ae*, *de*, *e* and *ade* are contained in the descriptions of objects g_1, g_2, g_3 and g_9. To generate less redundant patterns, we propose to replace arbitrary itemsets

g_1	a	b	c	d	e
g_2	a	b	c	d	e
g_3	a	b	c	d	e
g_4	a	b	c		
g_5	a	b	c		
g_6			c		
g_7	a	b			
g_8	a	b		d	
g_9	a			d	e
g_{10}	a				

Figure 4.5: A binary dataset

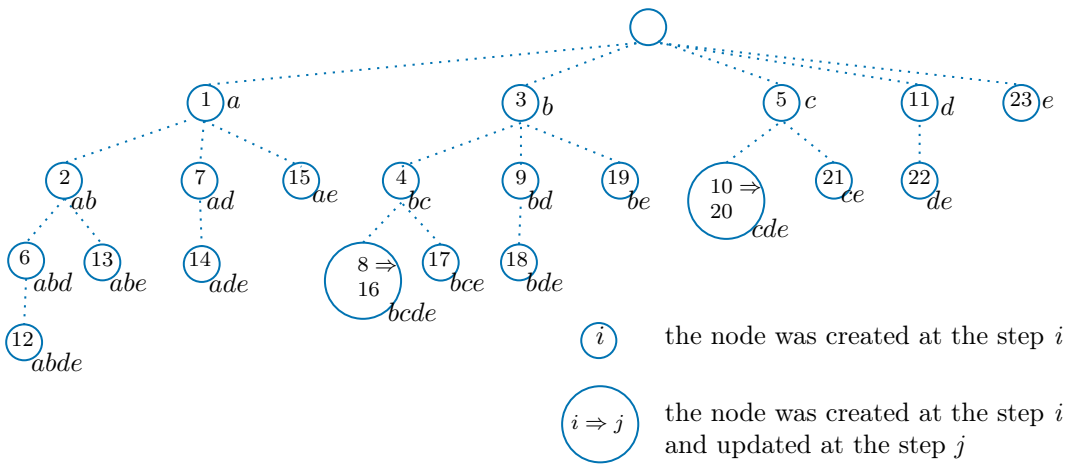


Figure 4.6: The execution tree of Algorithm 4 for dataset from Fig. 4.5

with closed ones.

Definition 18. *A closed itemset $X \subseteq M$ is called* likely-occurring closed *(LOC) if there exists $m \in X$ and $Y \subseteq X \setminus \{m\}$, $Y'' = X$ such that $P(X) > Q \cdot P(Y) \cdot P(\{m\})$.*

The definitions of LO and LOC (see Definition 16 and 18, respectively) are quite similar, however, an LOC itemset is not necessary LO. To be precise, an itemset is LOC if there exists an LO generator $Y \cup \{m\}$ of this itemset. By definition, this generator is not minimal.

In analogy to the relaxation of LO, we introduce a relaxation of LOC.

Definition 19. *Let $M^* = \{m_1, m_2, \cdots, m_k\}$ be a set of attributes arranged in order of decreasing frequency, i.e., $fr(m_i) \geq fr(m_j)$ for any $i \leq j$. A closed itemset X is* likely-occurring closed *(LOC) if there exists an LOC itemset $Y \subset X$ and $m \in X \setminus Y$ such that $fr(m) \geq \min_{m^* \in Y} fr(m^*)$, $X = (Y \cup \{m\})''$ and $P(X) > Q \cdot P(Y) \cdot P(\{m\})$.*

To compute LOC itemsets we use the same algorithm as for LO itemsets, its pseudocode is given in Algorithm 4. The only difference is in line 5 of the main algorithm. As for LO itemsets, the frequency constraints may be imposed.

Example. Let us consider how the algorithm works using the running example from Fig. 4.5. The corresponding execution tree is given in Fig. 4.7. The set of LOC itemsets contains much fewer itemsets than the set of LO itemsets. An LOC itemset is not necessarily LO, e.g., $m_1m_2m_3m_4m_5$ is not LO since $P(m_1m_2m_3m_4)P(m_5) > P(m_1m_2m_3m_4m_5)$, but it is LOC since $P(m_3)P(m_5) < P(m_3m_5)$.

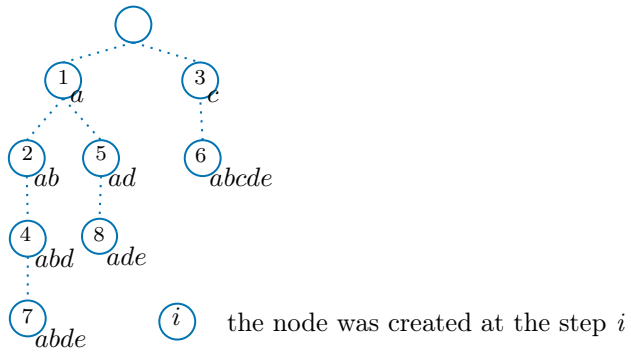


Figure 4.7: The execution tree of Algorithm 4 (for LOC itemsets) computed on the dataset from Fig. 4.5

4.3.2 Likely-occurring itemsets and related notions

Probability-based models are common in itemset and association rule mining. In this section we consider two widespread approaches to assess itemsets and association rules, and discuss how they are related to likely-occurring (closed) itemsets.

Independence model and lift. The models based on the comparison of estimated and observed probabilities of itemsets are quite common in the scientific literature [AH14]. The simplest model is the attribute independence model. Under this model, all items (attributes) are assumed to be independent. Attribute probability is approximated straightforwardly using the attribute frequency. Then, the probability of an itemset X is computed as follows:

$$P_{ind}(X) = \prod_{x \in X} P(x) = \prod_{x \in X} fr(x).$$

Despite its simplicity, this model is widely used in machine learning, e.g., Naïve Bayes classifiers are based on it. A natural extension of the attribute independence model is the partition independence model, where some partitions of X are assumed to be independent. Lift [BMUT97] is one of the most common measures to assess association rules under the partition independence model.

Definition 20. *Let $X \to Y$ be an association rule, then* lift *is given by*

$$lift(X \to Y) = \frac{P(XY)}{P(X)P(Y)} = \frac{fr(XY)}{fr(X)fr(Y)}.$$

Apart from lift, there is a lot of other measures (indices) that are based on the comparison of the antecedent and consequent supports, e.g., redundancy constraints [BTP+00, Zak00a], minimum improvement [BAG00], etc. They are commonly used to select association rules.

The notion of lift can be also adapted in different ways for itemset assessment. For example, one may assess the probability of an itemset X under the assumption that any two disjoint subitemsets, that make a partition of X, are independent. If the observed probability is greater than all the estimated probabilities obtained under this model, then the itemset is called *productive* [Web10].

The introduced above LO itemsets, in a certain sense, represent a particular case of productive itemsets. Instead of considering all possible partitions of X into two sets of items, we consider only one partition into $X \setminus \{m\}$ and m. Reformulating the definition of LO in terms of lift (for association rules), LO itemset X is an itemset that consists of LO itemset $X \setminus \{m\}$ and attribute m such that $lift(X \to m) > Q$, $Q \geq 1$. Since $X \setminus \{m\}$ is also LO, this reasoning can be done recursively. Hence, for an LO itemset X, there exists a sequence of the association rules $X_i \to m_i$ with growing in size antecedents $X_i = X_{i-1} \cup \{m_{i-1}\}$ and consequents m_i for which $lift(X_i \to m_i) > Q$.

Thus, if it is needed, one may reduce further the size of the discovered LO itemsets by using more tighter constraints, i.e., setting higher values for Q (in line 7 of the MERGE procedure given in Algorithm 4):

$$\frac{|(I_c \cup I_n)'|}{|G|} > Q \cdot \frac{|I'_c|}{|G|} \cdot \frac{|I'_n|}{|G|}.$$

The constraint above is equivalent to the constraint on lift of the association rule $I_n \to I_c$, i.e.,

$$lift(I_n \to I_c) = \frac{P(I_n \cup I_c)}{P(I_n) \cdot P(I_c)} > Q.$$

Moreover, because of the greedy strategy, the constraints hold recursively, i.e., there exist two disjoint subsets I^*_n, $I^*_c \subseteq I_n$ (for the LO itemsets there exists an additional constraint $I^*_n \cup I^*_c = I_n$) such that $lift(I^*_n \to I^*_c) > Q$. In experiments we consider how the proposed greedy strategy works for mining association rules on real-world datasets. Since the computing strategy is greedy, there are no guarantees that all LO/LOC itemsets (see Definitions 16 and 18, respectively) will be enumerated.

Itemset mining through compression Likely-occurring itemsets may be also useful for compression. In Section 4.2 we considered two MLD-based approaches for itemset mining called KRIMP and SLIM. Let us consider the relation between the itemsets selected by them and LO itemsets.

In KRIMP and SLIM, the length of an object encoded by itemsets from the code table CT is given by $length(g) = \sum_{X \in cover(g,CT)} L(code(X))$, where $cover(g, CT)$ is a subset of itemsets from CT used to cover (encode) the description $\{g\}'$ of object g.

Hence the compression is achieved by replacing several code words representing the itemsets that cover $B \subseteq \{g\}'$ with a single code word, representing B, such that $L(code(B)) < \sum_{X \in cover(B,CT)} L(code(X))$, where $cover(B, CT)$ is the cover of B by disjoint itemsets from the code table CT. The latter is equivalent to $\log P(B) > log(\prod_{X \in cover(B,CT)} P(X))$. Thus, we obtain the inequality $P(B) > \prod_{X \in cover(B,CT)} P(X)$, which is very similar to one from the definition of the LO itemsets.

Intuitively, in both cases, an itemset is considered optimal if its observed probability is greater than the estimated one. However, there are important differences between the models underlying the definition of "itemset optimality" (for the LO itemsets and the model used in KRIMP):

1. the both methods use different probability estimates of itemsets, namely, $P(X) = fr(X)$ (for the LO estimates) and $P(X) = \frac{usage(X)}{\sum_{Y \in \mathcal{P}} usage(Y)}$ (for the KRIMP-like models), where $usage(X)$ is frequency of X in the coverage, and $\mathcal{P}$ is the set of patterns;

2. the "optimality" of an itemset X in the compression-based models (KRIMP, SLIM) is evaluated not only w.r.t. the dataset but also w.r.t. the other patterns in the code table (i.e., adding of X should reduce the total description length).

Because of this discrepancy in the estimates, we cannot expect that the proposed method will work better than SLIM, where the itemset search space discovery is tailored to minimize the total description length. However, LOC itemsets may provide better results than the commonly used frequent closed itemsets (which are the candidates used by KRIMP). We compare different strategies for exploring itemset search space on real-world datasets in the next section.

4.3.3 Experiments

We use the discretized datasets from the LUCS-KDD repository [Coe03] and study LO and LOC itemsets[8] for two tasks, namely association rule and itemset mining.

[8]The source code for computing LO and LOC itemsets is available at `https://github.com/makhalova/pattern_mining_tools/blob/master/modules/binary/likely_occurring_itemsets.py`

Table 4.1: Parameters of datasets and the studied sets of itemsets

name	data description				closed itemsets			arbitrary itemsets			time (for itemsets)			#rules			
	$\|G\|$	$\|M\|$	density	#CL	#LOC	#FR.CL.	fr.thr.	#LO	#FR.	fr.thr.	LOC	LO	FR	LOC	FR.CL	LO	FR.
breast	699	14	0.64	360	74	74	0.33	283	287	0.41	0.01	0.02	0.12	4292	3980	7734	3972
ecoli	327	24	0.29	424	120	120	0.06	535	537	0.04	0.02	0.04	0.60	4768	2950	5418	9548
glass	214	40	0.22	3211	887	955	0.06	2478	2548	0.04	0.10	0.10	10.85	55262	18454	53948	62640
heart-dis.	303	45	0.29	25511	1928	1973	0.17	6390	6524	0.12	0.28	0.28	1.43	862252	22222	220560	324888
iris	150	16	0.25	106	45	47	0.05	72	83	0.05	0.00	0.01	0.03	274	320	352	372
led7	3200	14	0.50	1936	150	150	0.19	150	150	0.19	0.01	0.01	0.05	1484	1120	1484	1046
pima	768	36	0.22	1608	317	317	0.10	830	857	0.03	0.06	0.05	4.57	21294	7528	9044	22442
ticTacToe	958	27	0.33	42684	1880	1908	0.03	1900	1935	0.03	0.11	0.07	14.90	53816	13016	39696	13016
wine	178	65	0.20	13169	4914	5647	0.03	49719	59218	0.02	1.20	2.32	520.43	1771852	189378	4088770	3802456
zoo	101	35	0.46	4552	610	621	0.33	10344	11550	0.27	0.10	0.70	1.14	1609108	24736	1014508	1458272
average	**690**	**32**	**0.34**	**9356**	**1093**	**1181**	**0.14**	**7270**	**8369**	**0.12**	**0.19**	**0.36**	**55.41**	**438440**	**28370**	**544151**	**569865**

Association rule mining. Frequent (closed) itemsets are commonly used to mine association rules. In this section we study how useful LOC and LO itemsets are compared to frequent closed and frequent itemsets. In experiments we use 4 different sets of itemsets to compute rules: frequent closed (FR.CL.), likely-occurring closed (LOC), frequent (FR.), and likely-occurring (LO) itemsets. The itemsets are evaluated on 10 datasets, their parameters are given in Table 4.1. The number of objects and attributes is denoted by $|G|$ and $|M|$, respectively. The density of datasets (the ratio of 1's) is given in the column "density". The total number of closed itemsets is reported in the column "#CL". The total number of arbitrary itemsets has not been computed.

For each dataset we generate the whole set of LOC and LO itemsets ($Q = 1$, $F = 0$), the sizes of these sets are indicated in the columns "#LOC" and "#LO", respectively. In experiments we compare LOC itemsets with closed itemsets and LO itemsets with arbitrary itemsets. Since the number of closed and arbitrary itemsets is usually much higher than the number of LOC and LO itemsets, respectively, we limit the number of closed and arbitrary itemsets by choosing only frequent ones. The frequency threshold is chosen to ensure roughly the same sizes in the pairs of sets of itemsets under comparison. The frequency threshold for closed itemsets is chosen as the frequency of the $|LOC|$-th itemset in the list of closed itemsets ordered by decreasing frequency. It is indicated in the column "fr.thr." for closed itemsets. The frequency threshold varies a lot from dataset to dataset. For example, the smallest threshold is 0.06 for "ecoli" and "glass" datasets and the largest one is 0.33 for "breast" and "zoo" dataset. As we can see from the table, the sizes of "#LOC" and "#FR.CL." are quite close one to another but not necessarily equal because there are typically several itemsets with the same support size.

The frequency threshold for frequent arbitrary itemsets was chosen in the same way, i.e., as the frequency of the $|LO|$-th itemset in the list of arbitrary itemsets ordered by decreasing frequency.

To compute association rules we use a miner from MLxtent library implemented in Python[9]. The number of rules generated based on LOC, frequent closed (FR.CL.), LO, and frequent arbitrary (FR.) itemsets is reported in the column "#rules".

The number of rules generated based on the LOC itemsets is higher than the number of rules generated based on frequent closed itemsets. For example, for the "ecoli" dataset, the number of rules computed on 120 LOC and 120 frequent closed itemsets is 4768 and 2950, respectively. It can be explained by the fact that the size of the LOC itemsets is usually larger than the size of frequent closed itemsets. Thus, a larger amount of rules can be built on LOC itemsets by splitting each itemset into an antecedent and consequent. However, for non-closed itemsets, the number of rules generated based on LO itemsets may be lower than the number of rules generated based on frequent arbitrary itemsets. For example, for the "ecoli" dataset, where the number of LO and frequent itemsets is almost the same, the number of rules computed based on the LO itemsets is almost two times smaller than the number of rules computed based on frequent itemsets, i.e., 5418 vs 9548.

To evaluate their quality, we consider the most common quality measures for the association rules, namely *support*, *confidence*, *lift*, *leverage*, and *conviction*. We recall them below.

Let $X \to Y$ be an association rule with the antecedent X and the consequent Y, then the rule support

[9] `http://rasbt.github.io/mlxtend/`

is given by

$$support(X \to Y) = support(X \cup Y) = \frac{(X \cup Y)'}{|G|} \in [0, 1].$$

Confidence [AIS93] of a rule $X \to Y$ is the conditional probability of $X \cup Y$ given X. It is defined as follows:

$$confidence(X \to Y) = \frac{support(X \to Y)}{support(X)} \in [0, 1].$$

The maximal value is achieved when Y always occurs with X.

Lift [BMUT97] was discussed in the previous section. We recall it below. For a rule $X \to Y$ lift is given by

$$lift(X \to Y) = \frac{support(X \to Y)}{support(X) \cdot support(Y)} \in [0, \infty).$$

Leverage [PS91], like lift, is based on the comparison of the observed probability (frequency) of the rule and the probability estimated under the assumption that the antecedent and consequent are independent. Leverage is given as follows:

$$leverage(X \to Y) = support(X \to Y) - support(X) \cdot support(Y) \in [-1, 1].$$

For independent X and Y leverage is equal to 0.

The last measure we consider is conviction [BMUT97]. Like the measures above, conviction evaluates association rules under the independence model. However, unlike the previous measures that take into account co-occurrence of X and Y, conviction evaluates the independence of X and $\overline{Y}$:

$$conviction(X \to Y) = \frac{1 - support(Y)}{1 - confidence(X \to Y)} = \frac{support(X) \cdot support(\overline{Y})}{support(X \cup \overline{Y})} \in [0, \infty),$$

where $support(\overline{Y})$ is the number of transactions (objects) that do not contain Y. The high values of conviction indicate that the consequent depends a lot on the antecedent. Similar to lift, for independent X and $\overline{Y}$ the conviction values are equal to 1.

Let us proceed to the results of the experiments.

For the generated rules we consider mean values of the aforementioned quality measures as well as the 75th, 90th, and 95th percentiles. Considering the percentiles allows us to focus on the quality of the best itemsets, which are usually of interest to analysts. The averaged over 10 dataset values are reported in Fig. 4.8.

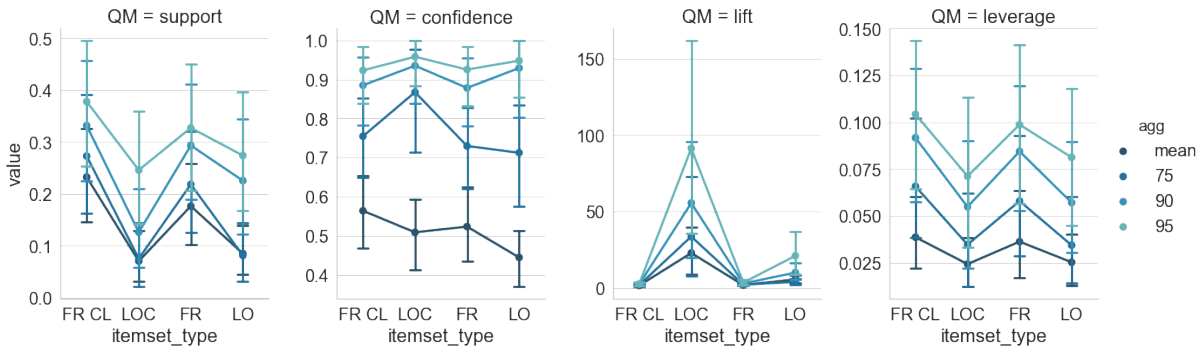


Figure 4.8: The averaged quality for 4 types of rules: computed based on frequent closed (FR.CL.), LOC, frequent arbitrary (FR.), and LO itemsets. The quality is measured by support, confidence, lift, and leverage. For each type of rules and each quality measure, the average values of mean, the 75th, 90th, and 95th percentiles over 10 datasets from Table 4.1 are reported

Since we do not set any frequency threshold for LOC and LO, the support of LOC- and LO-based rules, as expected, is lower than the support of the rules based on frequent and frequent closed itemsets (FR.CL.

and FR.). Regarding the confidence, the lowest mean values correspond to LO itemsets. However, the top $n\%$ of LOC- and LO-based rules have higher values than the top $n\%$ of FR.CL.- and FR.-based ones. For example, the top 10% values (the 90th percentile) of confidence are at least 0.935 and 0.929 for the LOC- and LO-based rules, and only 0.885 and 0.878 for FR.CL.- and FR.-based rules, respectively. Thus, considering the top rules, the LOC- and LO-based rules have better quality by confidence.

Regarding lift, LO- and LOC-based rules provide the best results. The difference in values is especially noticeable for the top 5% of rules (the 95th percentile). Top 5% LOC-rules have the highest values of lift, on average, 91.38. For the LO-based rules, the average value of lift for top 5% rules is 21.10, which is still greater than for FR.CL.- and FR.-based rules. However, the lift values of the top 5% of rules vary a lot from dataset to dataset (the standard deviation is shown in plots by horizontal lines). Nevertheless, the quality, measured by lift, is consistently higher for LO and LOC-based rules than for FR.CL.- and FR.-based rules.

The leverage is higher for FR.Cl.- and FR.-based rules. Despite the fact that lift and leverage differ only in the mathematical operations they use to compare the observed and estimated supports of rules and their parts, the analysis of rules based on these measures may lead to very different results. The high values of leverage (and low values of lift) for FR.CL.- and FR.-based rules are caused by a different order of magnitude of the supports. Very low supports (that is the case of LOC-based rules) result in high values of lift and low values of leverage.

For conviction, in some cases, we obtain infinite values, thus for each dataset we count the number of rules whose conviction value is equal to infinity as well as compute the average conviction value for the rules with finite values of this measure. The values are reported in Table 4.2. Our experiments show that, on average, the LOC and LO itemsets allow for generating a larger number of rules where the consequent highly depends on the antecedent (i.e., the conviction value is infinite). On average, the number of LOC-based rules with the infinite conviction is 179216, while for FR.CL.-based rules this number is 1192. For the majority of the studied datasets, the rate of the rules with the infinite values of conviction is larger for LOC-based rules than for the others. On average, the rate of LOC-/LO-based rules with infinite conviction is 22% and 12%, respectively, while the rate of FR.CL.-/FR.-based rules with infinite conviction is only 4% and 5%. The average finite conviction values are quite the same and do not exceed 2.

Table 4.2: Summary on conviction values

	#rules with inf. conviction values				rules with inf. conviction, %				average values of non-inf. conviction			
data name	FR.CL.	LOC	FR.	LO	FR.CL.	LOC	FR.	LO	FR.CL.	LOC	FR.	LO
breast	0	0	0	0	0	0	0	0	2.65	3.20	2.08	3.25
ecoli	114	1120	501	906	4	23	5	17	2.56	1.57	2.18	1.92
glass	473	16215	2477	8019	3	29	4	15	1.62	1.57	1.70	1.43
heart-dis.	349	297848	5090	30566	2	35	2	14	1.30	1.42	1.30	1.35
iris	42	41	39	44	13	15	10	12	2.12	2.10	2.10	1.97
led7	0	0	0	0	0	0	0	0	1.24	1.59	1.23	1.59
pima	0	5331	294	2189	0	25	1	24	1.19	1.21	1.04	1.15
ticTacToe	0	1836	0	390	0	3	0	1	1.07	1.17	1.07	1.14
wine	7849	972313	363391	788129	4	55	10	19	1.88	1.66	1.64	1.39
zoo	3088	497459	214319	205046	12	31	15	20	4.03	2.41	4.82	1.98
average	**1192**	**179216**	**58611**	**103529**	**4**	**22**	**5**	**12**	**1.97**	**1.79**	**1.92**	**1.72**

Thus, the analysis of the generated rules allows us to conclude that rules generated based on LO and LOC itemsets have better quality than the rules generated using roughly the same amount of frequent arbitrary and closed itemsets, respectively. The rules of the best quality are generated based on LOC itemsets.

Compression quality. In Section 4.3.2 we discussed the relation between LO itemsets and the itemsets ensuring good compression in Krimp and Slim. Since the estimates of LOC itemsets and itemsets selected by Krimp and Slim are based on different probability models, we cannot expect that the compression

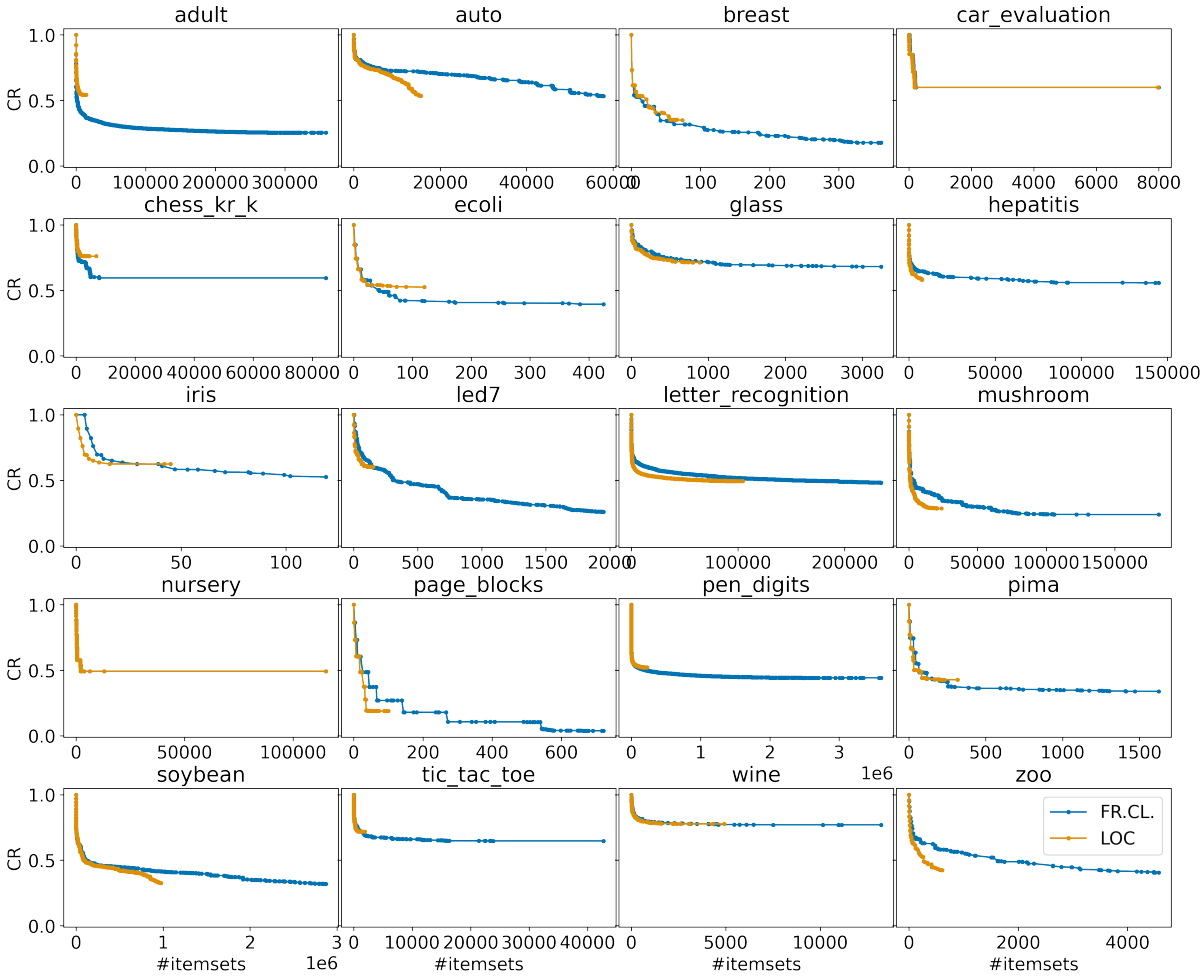


Figure 4.9: Compression quality of closed (FR.CL.) and LOC itemsets. The lower values are better

based on LOC itemsets will be more efficient that one based on the itemsets discovered by Slim since the estimates used in Slim conform with the used objective, while the probability estimates of LO itemsets do not.

In this section we compare the ability of LOC and closed itemsets to compress data. Both types of itemsets are computed independently from the data compression stage, in contrast to Slim, where itemsets are discovered gradually toward the direction ensuring better compression. Thus, a priori both LOC and closed itemsets compress worse than those discovered by Slim. Nevertheless, we study the applicability of LOC itemsets for this task and compare them with closed itemsets (used in the original version of Krimp. We emphasize that, in the compared approaches, the itemset search space is discovered independently of the itemset mining process.

To evaluate the ability of the itemsets to compress data, we consider how many itemsets we need to obtain a certain compression ratio. Fig. 4.9 shows how the compression ratio changes w.r.t. the number of considered itemsets. The initial state corresponds to the point (0,1), meaning that 0 itemsets have been used to compress data, and the compression ratio is maximal and equal to 1. The curves that are closer to the point (0,0) correspond to the best strategies of itemset search space discovery (i.e., the itemset set allows for compressing data better with a lower number of candidates). The experiments show that for "car evaluation", "wine" and "nursery" datasets the LOC itemsets do not provide any benefits over the closed itemsets. For the majority of datasets, the number of LOC is too small to ensure as good compression as with the whole set of closed itemsets, e.g., "adult", "breast", "led7", and others.

Table 4.3: Compression parameters for KRIMP based on closed itemsets, LO itemsets, and SLIM, where itemsets are discovered during the itemset mining process

type	CR			#candidates			time, seconds		
	KRIMP closed	KRIMP LOC	SLIM	KRIMP closed	KRIMP LOC	SLIM	KRIMP closed	KRIMP LOC	SLIM
adult	0.25	0.54	0.23	359141	14551	1521	77	39	51
auto	0.53	0.54	0.54	57788	15524	545	1	1	2
breast	0.18	0.35	0.16	361	74	38	0	0	0
car ev.	0.60	0.60	0.60	7999	7978	37	0	1	0
chess kr k	0.59	0.76	0.65	84635	6757	1226	46	15	2
cylinder bands	-	0.57	0.37	-	237887	540	-	2	2
ecoli	0.39	0.52	0.37	425	120	46	1	0	0
glass	0.68	0.71	0.65	3245	887	100	1	0	0
heart-disease	0.59	0.63	0.45	25538	1928	111	0	0	0
hepatitis	0.56	0.58	0.44	144870	7764	102	13	1	0
iris	0.53	0.62	0.53	119	45	25	0	0	0
led7	0.26	0.60	0.25	1950	150	110	3	0	0
letter rec.	0.48	0.49	0.32	234230	104546	2967	4281	171	131
mushroom	0.24	0.29	0.19	181945	23697	1433	30	5	14
nursery	0.49	0.49	0.49	115199	115172	341	38	11	1
page blocks	0.04	0.19	0.04	722	100	65	1	1	0
pen digits	0.44	0.52	0.39	3605506	228174	4106	9373	177	143
pima	0.34	0.43	0.31	1625	317	88	0	0	0
soybean	0.32	0.33	0.27	2874251	977469	730	116	19	1
tic tac toe	0.65	0.72	0.50	42711	1880	148	11	1	0
wine	0.77	0.78	0.77	13228	4914	144	2	1	0
zoo	0.41	0.42	0.37	4569	610	84	1	1	0
average	**0.44**	**0.53**	**0.40**	**369526**	**79570**	**659**	**666**	**20**	**15**

Among some of these datasets, we may still observe better behavior of LOC itemsets, e.g., for "hepatitis", "mushroom", "letter recognition", and "page blocks". There are also datasets where we achieve with the LOC itemsets as good compression as with closed ones but use a much lower number of candidates, e.g., "auto", "hepatitis", "soybean", "zoo".

Table 4.3 shows some compression characteristics of the studied methods, namely compression ratio CR, the number of considered candidates, and the running time of the compressors. The reported results show that SLIM is much more efficient than KRIMP run on closed and LOC itemsets. The running time of KRIMP can be too high, e.g., for "letter recognition", itemset mining with KRIMP on closed itemsets and LOC itemsets takes 4281 and 171 seconds, respectively, while the compression ratio is 0.48 and 0.49 when the closed itemsets and LOC itemsets are used, respectively. Thus, only a slight gain is achieved by KRIMP based on closed itemsets w.r.t. KRIMP based on LOC itemsets, while the running time in the first case is much higher than in the second case.

As we discussed in Section 4.3.2, LOC itemsets is not an optimal choice when a method for itemset search space discovery tailored for a chosen objective is available. The comparative study of KRIMP based on LOC itemsets and SLIM demonstrates it.

However, in general, when there is no specific algorithm for itemset search space discovery, LO and LOC itemsets may be quite useful.

In this section we considered an approach for discovering likely-occurring (LO or LOC) itemsets. This approach can be used for the discovery of both arbitrary and closed itemsets. Given a certain frequency threshold, this approach may also generate only frequent LO and LOC itemsets. In our experiments we show that the number of frequent enumerated LO/LOC itemsets is much lower than the number of frequent arbitrary and closed itemsets. However, with LO and LOC itemsets, we obtain association

rules of better quality. The proposed approach may be useful for compression as well, however, it does not outperform the methods where itemsets are discovered towards the direction minimizing the total description length.

4.4 Adapting the best practices of supervised learning to itemset mining

In the previous section we introduced the notion of likely-occurring itemsets and the algorithm for their generation. The algorithm can be used to discover an itemset search subspace in cases, where the standard frequency-based itemset enumerators return too many itemsets. In the experiments we demonstrated its particular usefulness for association rule mining.

In this section we focus on the problem of itemset mining in the narrow sense — computing a small set of interesting and non-redundant itemsets.

As it was discussed in Section 4.1.3, there exists a wide variety of approaches to mine interesting itemsets. Above, we considered in detail two of them, namely Krimp and Slim, which rely on a solid theoretical basis. However, in contrast with supervised learning, there is no universal quality criterion in pattern mining. The pattern mining methods use different interestingness measures and may rely on different quality measures to assess the results. A pattern set being the best w.r.t. one measure may not be the best w.r.t. another one.

In this section we address the problem of the lack of unified measures for evaluation of the results of pattern mining and propose to adopt the best practices of supervised learning to itemset mining. We rely on the assumption that those techniques that work well for building the models with an objective quality measure (in supervised settings) may provide good results when the ground truth is unavailable (in unsupervised settings).

4.4.1 KeepItSimple: an algorithm for discovering useful pattern sets

All the existing MDL-based approaches compute only one model, i.e., search for a set of jointly optimal (non-redundant) patterns. In [SK11], based on the assumption that a dataset may contain several models, the authors propose to compute a sequence of MDL-optimal pattern sets at different levels of granularity. They introduce a structural function over datasets. The structural function maps a natural number $k \in \mathbb{N}$ into a set of k MDL-optimal patterns. All pattern sets are computed independently of each other, and the model (pattern set) of size k may differ by more than one pattern from the model of size $k+1$. However, the patterns from different models (pattern sets) may be similar (or repetitive), thus, the whole set of models may be redundant.

The idea of using several models to describe data is widely used in supervised methods. The closest to itemset mining class of supervised learning methods is the one based on decision trees [Qui86]. A label of a tree node can be interpreted as an item of an itemset, thus an itemset is a set of labels in the path from the root to a leaf. All modern tree-based approaches are based on ensembles of trees, e.g., random forests [Bre01], extremely randomized trees [GEW06], boosted trees [FS95], etc.

The tree-based methods are known to have the following properties [MR14]: (i) an ensemble of trees is better than a single decision tree, (ii) an ensemble of "shallow" trees outperforms an ensemble of deeper trees (less prone to overfitting), (iii) an ensemble of boosted trees (an ensemble of trees where each new tree is focused on the examples misclassified by the previous trees) outperforms the ensembles where misclassified instances are treated in the same manner as those that are classified correctly. These rules of thumb work well in general, i.e., for a wide variety of datasets.

To the best of our knowledge, pattern mining approaches have never been considered from this perspective. Based on the assumption that the best practices of supervised learning may work well in unsupervised settings, we propose to adopt the aforementioned principles for building an ensemble of classifiers to itemset mining. As a basic method for computing a pattern set, we consider the state-of-the-art MDL-based itemset mining approach called Krimp [VVLS11]. The itemset sets computed by Krimp are similar to the decision trees since a decision tree can be represented as a set of itemsets (where each path from the root to a leaf corresponds to an itemset). However, there are two principal differences

	a	b	c	d	e
g_1	a	b	c		
g_2	a	b	c		
g_3	a	b			
g_4	a	b			
g_5	a			d	
g_6	a	b		d	e
g_7		b			e

$\mathcal{C}_1$	a	b	***abc*** (*ac*, *bc*)	***ad***	***be***	***abcde*** (*cd*, *ce*)
$\mathcal{C}_2$		***ab***	***abde*** (*ae*, *bd*, *de*)			

Figure 4.10: Dataset and the corresponding closure structure, biclosed itemsets are highlighted in bold. The pairs of items that generate the corresponding biclosed itemsets are given in parentheses

between the decision tree models and itemsets computed by Krimp: (i) the itemsets representing a tree make a partition while itemsets, selected by Krimp do not, and (ii) the absence (negation) of an item in tree-based models is considered as an item.

The proposed MDL-based multi-model approach (i) uses at most a quadratic number of "easily-derivable" candidates to optimal patterns, (ii) builds a sequence of models (pattern sets), where each successive model describes the data fragments that have been insufficiently well described by previous models, (iii) evaluates interestingness of patterns by their informativeness based on MDL. Non-redundancy of patterns within a single model (pattern set) is ensured by MDL, while non-redundancy of patterns across the models is ensured by the proposed projection strategy, i.e., focusing on poorly described data fragments to compute the next model.

In this section, we propose the KeepItSimple algorithm that exploits the ideas of the ensembles of simple models widely used in supervised learning and adapts it to itemset mining. We also revisit the MDL principle and propose so-called *variational description length* that allows for getting pattern sets of better quality.

In the next paragraphs, we present the principles of computing *simply-derivable itemsets* (an analog of shallow trees), *variational description length*, and then an approach for computing a *sequence of models* (pattern sets), where each new model (pattern set) describes those fragments of a dataset that are insufficiently well described by the previous models.

In experiments we show that proposed modifications improve the results of pattern mining w.r.t. very intuitive quality measures.

Simply-derivable patterns. One of the main shortcomings of pattern mining approaches is that they often compute candidates to optimal patterns in advance and then filter them (in contrast to the tree-based approaches, where a tree is built on the fly). Considering all frequent patterns as candidates is infeasible in practice and implies dealing with redundant (similar) patterns. Restricting the set of frequent patterns by increasing the frequency threshold does not solve this problem because often a relatively small number of frequent candidates does not ensure good compression. Moreover, some interesting and infrequent patterns might end up left out. Using only closed itemsets partially solves the redundancy problem, however, the number of closed itemsets may be still exponential.

We propose a deterministic threshold-free approach for building a candidate set containing at most a quadratic number of patterns w.r.t. the number of attributes. These patterns are "easily-derivable", since they are computed based on the closure of a union of two attributes, i.e., for a dataset D over attributes M, the set of candidates is given by $\mathcal{F} = \{\{m_k m_j\}'' \mid m_k, m_j \in M, k > j\}$, we call these patterns *biclosed itemsets*. It is worth noting that the set of biclosed itemsets is a subset of itemsets from the first two levels of the closure structure, introduced in Chapter 3.

Example. For the dataset from Fig. 4.10 (left), the set of biclosed itemsets is $\{ab, ad, be, abc, abde\}$. For biclosed itemsets of size at least 3 there are several ways to derive them, e.g., $ac'' = bc'' = abc$, $ae'' = bd'' = de'' = abcde$.

The biclosed itemsets are easily computable since it is enough to consider all pairs of attributes. However, the size of a biclosed itemset can be large, since we take the maximal (closed) itemset in the

equivalence class of itemsets composed of a pair of items. The number of biclosed itemsets is bounded from above by $|M|(|M|-1)/2$.

Variational description length. We also propose an alternative covering strategy that affects the total description length used in KRIMP: instead of covering only uncovered yet fragments of data, we cover all the fragments where a pattern appears.

On the one hand, it makes the pattern estimates less dependent on the greedy covering strategy, meaning that, in the case of new estimates, it does not matter in which order patterns are used to cover data. On the other hand, this covering is "redundant" and does not ensure the shortest code length. The latter means that, instead of deciding if a particular overlap allows for a shorter total description length, we allow for all possible overlaps. We cover even those fragments whose covering increases the total description length. This strategy seems irrational from the MDL perspective, however, it has a reasonable basis from the pattern mining perspective.

Let us consider the code tables from this perspective. The code table is a set of patterns and the associated (lengths of the) code words. In fact, the code word length indicates the importance of patterns w.r.t. a given dataset. In KRIMP, the code word length is inversely proportional to the pattern usage given in Equation 4.1. This probability, in turn, is based on the pattern usage in the covering. It means that the importance of patterns depends heavily on the order the patterns used to cover data. Thus, the estimates based on the usage may be unrelated to the occurrences of a pattern in the dataset and, in particular, they are not sensitive to multiple overlaps of patterns in the dataset.

According to the proposed covering strategy, we replace *usage* in Equation 4.1 with *frequency*. The probability estimates have the following form:

$$P(X) = \frac{frequency(X)}{\sum_{Y \in CT} frequency(Y)}. \tag{4.3}$$

It means that a pattern is not used anymore to *minimize* the total description length. Roughly speaking, for patterns in CT we compute the "pessimistic" total description length. Meaning that the encoding of data $L(D|CT)$ is maximal, however, the length of the code table $L(CT)$ is, in a certain sense, "optimal" w.r.t. the maximal usage of patterns in the data covering.

It is worth noting that the set of patterns constituting the code table is computed gradually for both types of estimates, thus, in both cases, we keep a pattern only if it allows for a shorter total description length. Thus, even with the proposed "redundant" covering we minimize the total description length, but this total length is estimated in a "pessimistic" way.

Even though the proposed modification violates the compression principle, it fits the pattern mining intuition: (i) each overlapped region contributes to the total description length (moreover, we imposes the heaviest penalty on similar patterns), (ii) the pattern estimates are not affected by the chosen covering strategy. The latter means that given a set of patterns with the associated code word lengths, we do not need anymore to refer to the greedy covering strategy to explain why a particular pattern is more important than the other one. Using the frequency-based estimates, we rely on a very intuitive reasoning w.r.t. the dataset rather than on the covering strategy.

Example. In Fig. 4.11 and Fig. 4.12 we show the difference in the covering when the total length is minimized using the standard covering strategy (from KRIMP) and the proposed one. In fact, the probability estimates in both cases are the same (see Equation 4.1), but for the proposed covering strategy "usage" is equal to "frequency". As candidates to optimal patterns we consider itemsets *ab*, *abc*, *ad*, and *be*. They are added in the same order in both cases. In Fig. 4.11 we use the standard (disjoint) covering, as in KRIMP. In Fig. 4.12 we use the proposed covering strategy with overlaps.

At the first step, both coverings are the same. However, at the second step, itemset *abc* is accepted as optimal for disjoint covering, and discarded for the covering with overlaps. At the third and fourth steps, *ad* and *be* are accepted to the code table only for the proposed strategy (with overlaps). Thus, the set of MDL-selected patterns computed using the standard covering strategy is $\{abc, ab\}$, while the set of optimal patterns w.r.t. the proposed strategy is $\{ab, ad, be\}$.

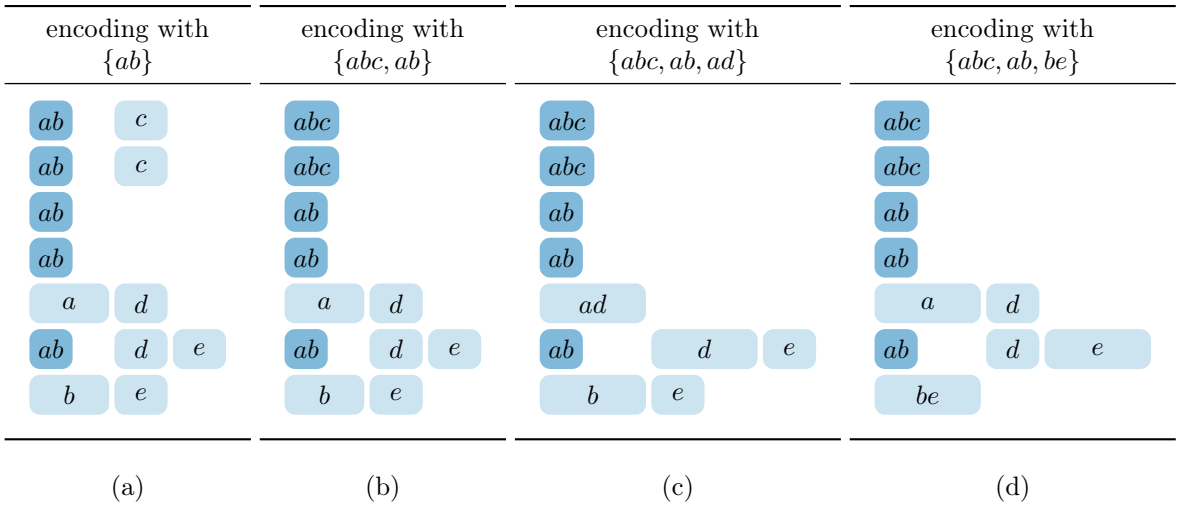


Figure 4.11: The standard covering: (a) adding ab (optimal), (b) adding abc (optimal), (c) adding ad (not optimal), (d) adding be (not optimal). The set of selected itemsets is $\{abc, ab\}$, the corresponding encoding is given in (b)

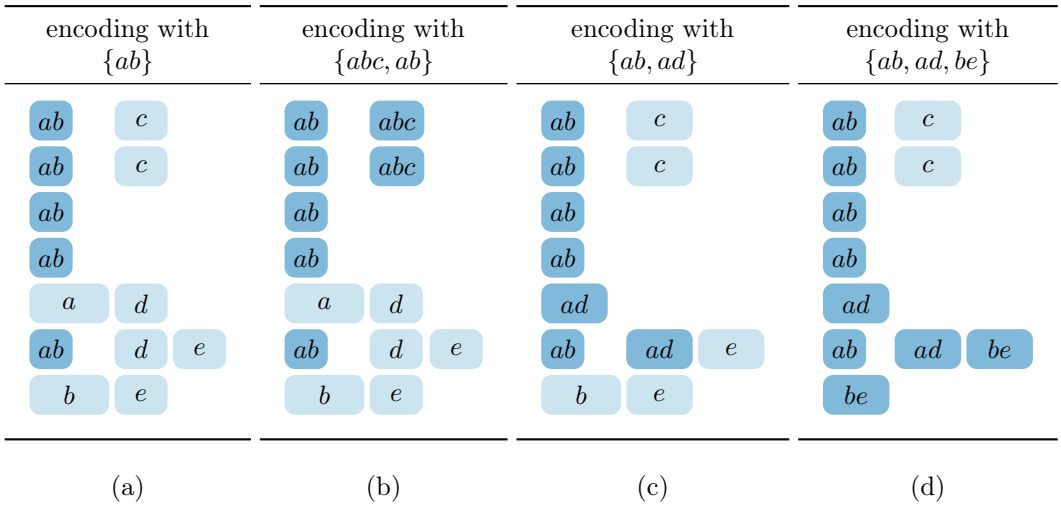


Figure 4.12: The modified covering: (a) adding ab (optimal), (b) adding abc (not optimal), (c) adding ad (optimal), (d) adding be (optimal). The set of selected patterns is $\{ab, ad, be\}$, the corresponding encoding is given in (d)

Multi-model description. Since the number of biclosed candidates is much lower than the number of frequent patterns, they do not cover a dataset as well as an exponentially large pattern set, e.g., like closed itemsets used in Krimp. Thus, larger fragments of data may remain uncovered. Using the principle of "boosting", i.e., building a new model based on the data fragments that have been poorly described by the previous models, we propose to build a data description gradually by a sequence of models (pattern sets). At each iteration, we consider a data fragment (projection) U of the binary dataset D that has not been entirely described by the models from the previous iterations. We propose an algorithm, called KeepItSimple (KIS), its pseudocode is given in Algorithm 5. We present two versions of KIS, KIS-S and KIS-D, that differ in the way of computing projections (i.e., defining which fragments are insufficiently well described). S stands for the simple projection and D stands for the detailed projection.

Algorithm 5 KEEPITSIMPLE(D)

Input: binary dataset D over set of attributes M
Output: itemset collection $\mathcal{P}$

```
 1:  L ← StandardCodeTableLength(D)
 2:  U ← D; I ← M
 3:  F ← ComputeBiclosed(U)
 4:  L_old ← L + 1
 5:  while (L_old > L) and (I ≠ ∅) do
 6:      L_old ← L
7S:      F ← ComputeNewBiclosed(F, I)
7D:      F ← ComputeBiclosed(U)
 8:      C ← MDLOptimal(U, F)
 9:      L ← L(U | C) + L(C | U)
10:      U ← U \ Cover(U, C)
11:      I ← UncoveredAttributes(U)
12S:     U ← Projection(D, I)
12D:     U ← Projection(U, I)
13:      P ← P ∪ C
14:  end while
15:  return P
```

KIS-S

Step 1

$\mathcal{F} = \{ab, abc, ad, be\}$
$\mathcal{C} = \{ab, abc\}$
$Cover(U, \mathcal{C})$:

g_1	a	b	c		
g_2	a	b	c		
g_3	a	b			
g_4	a	b			
g_5	a			d	
g_6	a	b		d	e
g_7		b			e

Step 2

$\mathcal{F} = \{ad, be\}$
$\mathcal{C} = \{ad, be\}$
$Cover(U_S, \mathcal{C})$:

g_1	a	b		
g_2	a	b		
g_3	a	b		
g_4	a	b		
g_5	a		d	
g_6	a	b	d	e
g_7		b		e

Step 3

$\mathcal{F} = \emptyset$
$\mathcal{C} = \emptyset$
$Cover(U_S, \mathcal{C})$:

g_1	a	b
g_2	a	b
g_3	a	b
g_4	a	b
g_5	a	
g_6	a	b
g_7		b

KIS-D

Step 1

$\mathcal{F} = \{ab, abc, ad, be\}$
$\mathcal{C} = \{ab, abc\}$
$Cover(U, \mathcal{C})$:

g_1	a	b	c		
g_2	a	b	c		
g_3	a	b			
g_4	a	b			
g_5	a			d	
g_6	a	b		d	e
g_7		b			e

Step 2

$\mathcal{F} = \{\}$
$\mathcal{C} = \{\}$
$Cover(U_D, \mathcal{C})$:

g_1				
g_2				
g_3				
g_4				
g_5	a		d	
g_6			d	e
g_7		b		e

Figure 4.13: The steps of the KEEPITSIMPLE algorithm for the dataset form Fig. 4.10

To compute a single model we use the KRIMP algorithm with two modifications presented above: (i) instead of frequent (closed) patterns, we use biclosed itemsets as candidates, (ii) instead of disjoint covering, we use the covering with all possible overlaps (that results in the probability estimates given in Equation 4.3).

KIS starts with the standard code table (line 1), the candidates $\mathcal{F}$ are biclosed itemsets (line 3). Then candidates from $\mathcal{F}$ are used to cover greedily the dataset. When the first pattern set (model) is computed, there are two possibilities to define projections, i.e., the dataset U that will be used instead of D at the next iteration to compute a new set of optimal patterns. We consider a *simple* (S) and *detailed* (D) projections. To compute the projection (line 12), we consider the set of attributes $I \subseteq M$ that are partially covered by patterns from the current code table. To compute S-projection, we keep the whole columns corresponding to the partially covered attributes, while to compute D-projection we keep only uncovered fragments in these columns (see line $12S$ and $12D$, respectively). The process of computing new tables stops when there are no new biclosed patterns in the set of selected ones, or the next optimal code table is the standard one.

Example. Let us consider how the KIS algorithm works using the dataset from Fig. 4.10. As candidates we consider biclosed itemsets having support at least 2. The steps of Algorithm 5 are given to the right of the pseudocode. To demonstrate the difference between the projections, we use the standard covering strategy used by Krimp, since using KIS returns the same set of itemset, when the proposed cover strategy is used. At the first step we use the original dataset D instead of projections. The set of biclosed itemsets is $\mathcal{F} = \{ab, abc, ad, be\}$. The set of selected patterns is $\mathcal{C} = \{ab, abc\}$. For covered data $Cover(U, \mathcal{C})$ the projections U_S and U_D are given in Step 2. For S-projection, the whole columns containing at least one uncovered item are used (namely "a", "b", "d", "e"), while for D-projection we use only uncovered fragments. These projections are used to compute new candidates. The set of biclosed itemsets computed on U_S at Step 2 is $\mathcal{F} = \{ad, be\}$. The set of biclosed itemsets computed on U_D is empty (since ad and be represent single objects and have support 1). The selected patterns are $\mathcal{C} = \{ad, be\}$ and $\mathcal{C} = \emptyset$, for S and D-projections, respectively. Step 3 is performed only for the S-projected data. Since the only possible biclosed itemset ab is already in the set of selected patterns, the candidate set $\mathcal{F}$ is empty. The sets of selected patterns are $\mathcal{C} = \{ab, abc, ad, be\}$ and $\mathcal{C} = \{ab, abc\}$, for KIS-$S$ and KIS-D, respectively.

In the next section we present the results of the experimental evaluation of KeepItSimple and discuss the benefits of KeepItSimple w.r.t. Krimp.

4.4.2 Experiments

In this section we report the results of an experimental study. We compare Krimp [VVLS11] (disjoint covering) with KeepItSimple. In KeepItSimple we use both the covering strategy used in Krimp and the proposed one (with overlaps). For KeepItSimple[10] we study simple and detailed projections.

We evaluate the methods on freely available datasets (*adult*, *auto*, *breast*, *car evaluation*, *chess*, *ecoli*, *glass*, *heart disease*, *hepatitis*, *iris*, *led 7*, *mushroom*, *nursery*, *pima indians*, *soybean large*, *tic-tac-toe*, *wine*, *zoo*) from the LUCS/KDD discretized data set repository [Coe03].

As the quality measures, we consider the following ones.

Size of pattern sets. Small pattern sets can be easily checked by experts. Thus, the smaller sizes are preferable.

Descriptiveness. We define the coverage ratio as the ratio of entries that are covered by itemsets to all non-empty entries. An itemset set is more descriptive if it covers a larger portion of data. Formally speaking, the coverage ratio is defined as follows:

$$CoR(\mathcal{P}, D) = 1 - UR(\mathcal{P}, D),$$

where the uncovered cell ratio $UR(\mathcal{P}, D)$ is defined in Equation 2.3.

Redundancy (overlapping rate) is the average number of itemsets that cover an entry. For non-redundant itemset sets, this value is close to 1. The high values indicate redundancy, i.e., an entry is described by several itemsets. The overlapping rate is introduced in Section 2.4, Equation 2.2.

Interestingness (F-measure, precision, recall). In this study we consider interestingness as the ability of patterns to describe "hidden interesting groups of objects". To evaluate interestingness we take into account the class labels of objects. We use these class labels only to evaluate patterns. As a quality

[10]The source code of KeepItSimple is available at `https://github.com/makhalova/pattern_mining_tools/blob/master/modules/binary/keep_it_simple.py`

measure we choose *F-measure* (F_1 score) since it takes into account both precision and recall of a set of objects described by a pattern:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

The results of the experiments are given in Fig. 4.14. The bars/dashes correspond to the averaged values/standard deviation, respectively. The results show that KIS provides more stable results, i.e., the standard deviation is smaller for the first three characteristics, namely the number of optimal patterns, coverage ratio, and the overlapping rate. The quality of patterns (evaluated with F_1 measure) is higher for KIS (any settings) than for Krimp. Further, we compare approaches in more detail.

Models with disjoint estimates. Let us compare Krimp and KIS based on S- and D-projections, where to estimate pattern probability, we use Equation 4.1 (i.e., the "disjoint" covering strategy). The results correspond to orange bars in the figure. Among KIS-S, KIS-D, and Krimp, KIS-S has the largest average number of patterns, the highest coverage ratio, and the largest number of patterns per cell. That allows us to conclude that KIS-S-optimal pattern sets are the most "redundant". However, having a larger number of patterns, KIS-S-optimal pattern sets describe better the datasets.

KIS-D generates roughly the same number of patterns as Krimp and has almost the same average overlapping rate (redundancy), but KIS-D itemsets describe larger fragments of data than Krimp.

From these results we conclude that replacing frequent closed itemsets with biclosed ones and using a multi-model description allows us to improve the descriptiveness (coverage ratio) of pattern sets. In the case of KIS-D, the "complexity" of pattern sets (i.e., # patterns, overlapping rate) does not increase w.r.t. Krimp. Moreover, the characteristics computed for KIS-D-generated itemset sets are more stable (i.e., the standard deviation over the studied datasets is lower than for Krimp-generated itemsets).

Models with overlapping estimates. The probability estimates from Equation 4.3 were proposed to improve the redundancy of patterns. Let us compare Krimp with the KIS algorithm (both S- and D-projections), where in KIS we use the new estimates. The results corresponding to the itemset sets computed by KIS with the new estimates are shown in the figure in the blue bars. The results show that KIS returns smaller in size pattern sets (w.r.t. Krimp) that describe larger data fragments and have lower overlapping rates. Thus, with the proposed estimates we get more descriptive models (by coverage ratio) that contain a smaller number of patterns. It is important to notice that the quality of patterns (assessed by F_1) for these pattern sets is higher than that for the Krimp-selected ones.

For KIS we also consider the average number of code tables (models) computed using two types of estimates. On average, the number of KIS-D-generated code tables is lower (2.69 and 2.54 for "standard" and "proposed" estimates, respectively). The average number of KIS-S-generated models is 4.08 and 3.54 for "standard" and "proposed" estimates, respectively. Thus, KIS-S converges faster than KIS-D, i.e., it needs a lower number of code tables to compute the whole set of patterns.

To sum up, the proposed modifications, namely (i) (simply-derivable) biclosed itemsets, (ii) a multi-model description, and (iii) new probability estimates, allow us to improve the results of pattern mining w.r.t. Krimp by very intuitive quality measures. Depending on chosen settings, we can significantly improve a particular quality measure. The best overall results are achieved by KIS based on D-projections with the proposed estimates (allowing overlaps). Applying KIS with these settings allows us to reduce considerably the size of pattern sets and pattern redundancy, improving descriptiveness and "interestingness" of patterns.

4.5 Discussion and conclusion

In this chapter we considered the problem of itemset and association rule mining, where the notion of interestingness is fundamental. However, this notion is subjective. As a consequence, there exists a large variety of methods and approaches to define interestingness, and the results obtained by these methods may be very different. The lack of ground truth complicates the process of learning from data. That is why the background knowledge or even intuition of experts often play a crucial role in practice.

The evaluation of the results of pattern mining is also quite complicated since the evaluation of a subjective notion leaves room for options: a pattern may be interesting for a particular expert or may

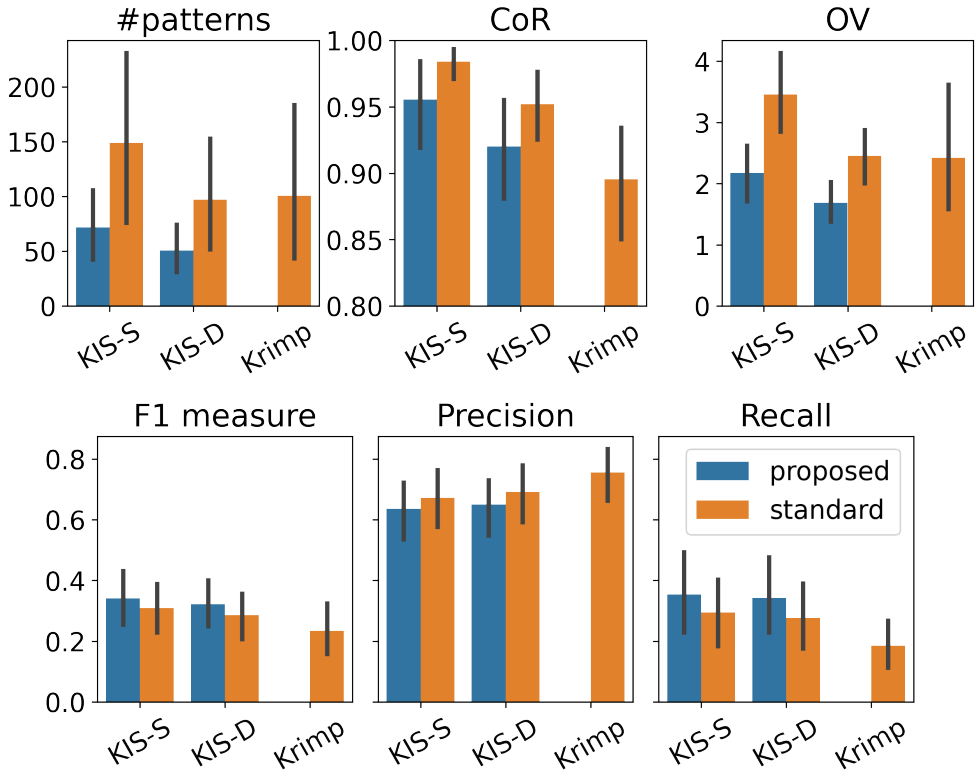


Figure 4.14: The average values (bars) and standard deviation (dashes) of the quality measures of pattern sets computed by KIS (S-, D-projections) and KRIMP. For *the number of patterns*, we exclude the results for the "adult" dataset since the number of KRIMP-generated patterns is drastically higher than the number of KIS-S- and KIS-D-generated ones with the same standard estimates of the pattern probability (1283 vs 639 and 497, respectively)

bring interesting knowledge to the whole domain. A lot of modern approaches are based on the MDL principle, which is often referred to as *learning by compression*. Put differently, the patterns are considered to be interesting if they compress well. Thus, in the MDL-based approaches, the problem of mining interesting patterns is reduced to searching some regularities in data allowing for good compression.

However, apart from evaluating interestingness (by *interestingness measures*), we take into account *quality measures* that formalize very intuitive expectation from the results, i.e., non-redundancy, good descriptiveness, compactness, etc.

In this chapter, we addressed some important issues that follow directly from the aforementioned:

– pattern explosion problem;

– compliance of the interestingness measure with the quality measures.

Pattern explosion is a common problem for pattern mining approaches. In this section we proposed an approach for reducing the itemset search space relying on a variation of the independence model and assuming that a pattern is interesting if it appears more often than it was expected. We emphasize that this approach may replace frequency-based enumeration algorithms, however, if there exists an algorithm that is tailored for the specific interestingness measure, it is preferable to use it.

Regarding the second issue — compliance of the interestingness measure with the quality measures — we revisit the problem of the lack of ground truth in pattern mining. Based on the assumption that the best technique from supervised learning with the available ground truth, which work generally well, may work well even when no ground truth is available.

We propose the KeepItSimple algorithm that exploits some ideas that are widely used for constructing the decision tree ensembles: biclosed itemsets (an analog of shallow trees and stumps), multi-model description with the projections (an analog of boosted trees), and modification of the cover function from [VVLS11] (for the conformity of the interestingness measure with the quality measure and to make the itemset estimates less dependent on the heuristics). Our experiments shows that KeepItSimple improves the results of Krimp w.r.t. very intuitive quality measures.

In this chapter we showed that the existing methods for itemset mining with a strong theoretical foundation can be modified with some heuristics reflecting our intuition or with common techniques from supervised learning. The latter demonstrates great potential for developing methods involving background knowledge or the best practices from the other fields to obtain better results of pattern mining.

5

Pattern mining in numerical data

Contents

5.1 Introduction **83**
5.2 Related work **84**
5.2.1 Numerical data preprocessing 85
5.2.2 MDL-based approaches to pattern mining 86
5.3 Basic notions **87**
5.3.1 Formalization of data and patterns 87
5.3.2 Information theory and MDL 88
5.3.3 Derivation of the plug-in codes 90
5.3.4 ML-estimates of the parameters of the multinomial distribution 90
5.4 Mint **91**
5.4.1 The model encoding 91
5.4.2 The Mint algorithm 93
5.5 Experiments **97**
5.5.1 Datasets 97
5.6 Discussion and conclusion **113**

Pattern mining is well established in data mining research, especially for mining binary datasets. Surprisingly, there is much less work about numerical pattern mining and this research area remains under-explored. In this chapter we propose Mint, an efficient MDL-based algorithm for mining numerical datasets. The MDL principle is a robust and reliable framework used in pattern mining, and as well in subgroup discovery. In Mint we reuse MDL for discovering useful patterns and returning a set of non-redundant overlapping patterns with well-defined boundaries and covering meaningful groups of objects. Mint is not alone in the category of numerical pattern miners based on MDL. In the experiments presented in the chapter, we show that Mint outperforms competitors among which IPD [NMVB14], RealKrimp [WDKG14], and Slim [SV12].

5.1 Introduction

As it was discussed in the previous chapter, the objective of pattern mining is to discover a small set of interesting patterns that describe together a large portion of a dataset and can be easily interpreted and reused. We discussed above that pattern mining encompasses a large variety of algorithms in knowledge discovery and data mining aimed at analyzing datasets [VT14]. Present approaches in pattern mining are aimed at discovering an interesting *pattern set* rather than a set of individually interesting patterns, where the quality of patterns is evaluated w.r.t. both the dataset and other patterns. One common theoretical basis for pattern set mining relies on the minimum description length principle, which is applied to many

types of patterns, e.g. itemsets [VVLS11], patterns of arbitrary shapes in 2-dimensional data [FvL20], sequences [TV12c], graphs [BCF20], etc.

Contrasting the recent advances in pattern mining, algorithms for mining numerical data appear to be insufficiently explored. To date, one of the most common ways to mine numerical pattern sets relies on the application of itemset mining to binarized datasets. This is discussed below in more detail but before we would like to mention an alternative to numeric pattern mining which is clustering.

In the last few decades, clustering algorithms have been extensively developed, and many different and efficient approaches have been proposed [Jai10, vCDB17, JMG20]. However, there is an important conceptual difference between pattern mining and clustering. In pattern mining the *description* comes first, while in clustering the primacy is given to the *object similarity*. In other words, numerical pattern mining is more interested in the description of a group of objects in terms of a set of attributes related to these objects, while clustering focuses more on the detection of these groups of objects based on their commonalities as measured by a similarity or a distance. The former entails some requirements for the ease of interpretation, i.e., the resulting patterns should describe a region in the "attribute space" that is easy to interpret. By contrast, in clustering, the focus is put on groups of objects or instances. The clusters can be constrained to have certain shapes, e.g., spheres in K-means or DBSCAN, but still the similarity of objects remains the most important characteristic of clusters. For example, clustering techniques such as agglomerative single-linkage clustering in a multidimensional space may return clusters of very complex shapes. Usually no attention is paid to these shapes while this is one of the most important preoccupations in numerical pattern mining.

Accordingly, in this chapter, we propose an MDL-based approach to numerical pattern set mining called Mint for "Mining INTeresting Numerical Pattern Sets". Mint computes numerical patterns as m-dimensional hyper-rectangles which are products of m intervals, where the intervals are related to the attributes and their values. The main benefits of the Mint approach are that (i) Mint does not need to explore the pattern space in advance as candidates to become optimal patterns are computed on the fly, (ii) the total number of explored patterns is at most cubic (and it is at most quadratic at each iteration) in the number of objects with distinct descriptions considered as vectors of attribute values, (iii) Mint is based on MDL and outputs a small set of non-redundant informative patterns.

In addition, a series of experiments shows that Mint is efficient and outputs sets of patterns of high quality: the patterns describe meaningful groups of objects with quite precise boundaries. Actually, the Mint algorithm is able to mine numerical patterns both on small and on large datasets, and it is most of the time more efficient and reliable than its competitors Slim and RealKrimp. The proposed encoding scheme is based on prequential plug-in codes, which have better theoretical properties than the codes used in Slim, RealKrimp and an MDL-based method for discretization called IPD.

The chapter has the following structure. In Section 5.2 we discuss the state-of-the-art algorithms in itemset pattern mining for numerical data. Section 5.3 introduces the new notions used in the chapter, while in Section 5.4 we describe the bases of the proposed method. Next, Section 5.5 relates the experiments carried out for illustrating the good behavior and the strengths of Mint. Finally, Section 5.6 concludes the chapter with a discussion about the potential of Mint and some directions for future work.

5.2 Related work

The problem of pattern mining has been extensively studied for binary data — itemset mining — but remains much less explored for numerical data. Hence a common way to mine patterns in numerical data relies on a binarization of data and then application of itemset mining algorithms. Meanwhile, a number of approaches was designed for mining numerical datasets possibly involving binarization and taking into account the type of the data at hand. In this section we firstly discuss different numerical data preprocessing approaches allowing the use of itemset mining and then we discuss state-of-the-art approaches in numerical pattern mining.

5.2.1 Numerical data preprocessing

The data preprocessing is the cornerstone for discovering patterns of good quality and relies on *discretization* or *binarization* tasks.

Discretization. Discretization relies on partitioning the range of an attribute value into intervals and then mapping the intervals into integers for preserving the order of the intervals. The existing discretization techniques can be categorized into *univariate* and *multivariate* techniques.

Univariate discretization includes all the methods where attributes are considered independently. An attribute range may be split into intervals of equal width or equal height w.r.t. frequency. Another way to split an attribute range is based on the K-MEANS algorithm [DPD11], where some criteria are used for assessing the quality of clustering and choosing an optimal K. A more flexible approach consists in splitting based on the MDL principle [KM07, RSY92] which is discussed in more length below.

However, considering each attribute range independently does not preserve the interaction between attributes and, as a consequence, may make some patterns not recognizable. Multivariate discretization techniques were proposed to tackle this issue. In [MPY05, KWL+06], multivariate discretization is based on principal component analysis and independent component analysis, respectively. However, both techniques do not guarantee to take into account possible complex interactions between attributes and require some assumptions on either distribution or correlation [NMVB14]. Accordingly, in [NMVB14] an MDL-based algorithm for multivariate discretization was proposed. This algorithm shows the shortcomings mentioned above and which works in unsupervised settings (contrasting a related approach in [FI93, Bou06, BBL10]). Indeed, MDL is used in a large number of approaches and is detailed below in Section 5.2.2.

Binarization. Discretization is not the only step to accomplish before applying an itemset mining algorithm. Another important operation is binarization, i.e., the transformation of discrete values into binary attributes. Binarization should be carefully taken into account as it may affect the results of itemset mining and induce the loss of important information. Moreover, binarization is associated with the risk of introducing artifacts and then obtaining meaningless results.

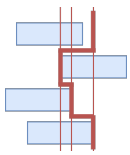
A simple and popular binarization is based on "one-hot encoding", where each discrete value is associated with one binary attribute. The number of attributes may become very large which can lead to an excessive computational load. Moreover, one-hot encoding does not necessarily preserve the order of discrete values.

By contrast, interordinal scaling preserves the order of values by introducing for each discrete value v two binary attributes "$x \geq v$" and "$x \leq v$". However, in [KKN11b] it was shown that, with a low-frequency threshold, mining itemsets in interordinally scaled data becomes much more expensive than mining hyper-rectangles in numerical data. Hyper-rectangles here are studied in the framework of interval pattern structures that were introduced in Section 2.2.

An alternative approach to interordinal scaling [SA96] consists in computing more general itemsets based on considered discrete values. The authors introduce the notion of the partial completeness w.r.t. a given frequency threshold. This notion allows one to formalize the information loss caused by partitioning as well as to choose an appropriate number of intervals w.r.t. chosen parameters. Like interordinal scaling, this approach suffers from pattern explosion. In addition, this method requires to set some parameters, e.g., frequency threshold and completeness level, however, their optimal values are unknown.

Despite its limitations, one-hot encoding remains a good option provided that suitable constraints can be defined for avoiding attribute explosion and allowing for tractable computation.

Numerical attribute set assessment based on ranks. One of the main drawbacks of the above-mentioned approaches, which consider discretization and binarization as mandatory preprocessing steps, is that the quality of the output depends on the quality of the discretization. In mining numerical patterns, when the boundaries of patterns are not well aligned as shown in the figure on the right, uniform boundaries will produce imprecise descriptions, while using exact boundaries may greatly complicate pattern mining.



An alternative approach that "simulates" multi-threshold discretization is proposed in a seminal paper [CGJ06]. It consists in (i) considering the ranks of attribute values instead of actual real values, and (ii) evaluating the sets of numerical attributes using rank-based measures. In [CGJ06], the authors propose several support measures based on ranks. In such a way, the problem of dealing with attribute values themselves is circumvented by considering the coherence of the ranked attribute values. Moreover, in [Tat13], Tatti proposes two scores to evaluate a set of numerical attributes using ranked attribute values as well. The scores are used to find the best combinations of attributes w.r.t. rank-based supports.

In all the methods mentioned in this subsection, patterns are understood as combinations of the attribute ranges as a whole. These methods do not provide descriptions related to some particular parts of the range if needed, which is the main focus of this chapter.

5.2.2 MDL-based approaches to pattern mining

The MDL approach to pattern set mining is based on the slogan: "the best set of patterns is the set that compresses the database best" [Grü07]. There is a significant amount of papers about the use of the MDL principle in pattern mining and this is very well presented in the report of Galbrun [Gal20]. MDL has been used in many different contexts but hereafter we focus on pattern mining. In the previous chapter we focused on specific approaches to itemset mining called Krimp and Slim. Krimp is one of the most famous MDL-based itemset miners, which was introduced in [VVLS11]. Krimp relies on two steps that consist in (i) generating a set of frequent patterns, and (ii) selecting those minimizing the total description length. While Krimp is an efficient and well-designed itemset miner, it requires that all frequent itemsets should be generated. Moreover, increasing the frequency threshold may lead to a worse compression, so the Slim system [SV12] was proposed to tackle this issue. In contrast to Krimp, Slim does not require that all frequent closed itemsets should be generated in advance, since the candidates to become selected itemsets are discovered gradually (for details see Section 4.2). Nevertheless, the encoding scheme used in Krimp and Slim shows a range of limitations that are discussed in more detail in Section 5.3.2. In continuation, the DiffNorm algorithm [BV15] is an extension of Slim that is based on a better encoding scheme and can be applied to a collection of datasets for finding the difference in datasets in terms of itemsets. Another MDL algorithm related to the Krimp family was proposed in [ATVF12] for fixing the scalability issues. This algorithm deals with categorical data and is less sensitive to combinatorial explosion.

All the aforementioned MDL-based algorithms represent a "model", i.e. a set of patterns, as a two-column table, where the left-hand column contains the pattern descriptions and the right-hand column contains the associated code words. Another way to store patterns is proposed in the Pack algorithm [TV08], where the model is encoded as a decision tree, so that a node corresponds to an attribute. A non-leaf node has two siblings reflecting the presence or absence of this attribute in an itemset. The itemset, in turn, is a path from the root to a leaf node. One main difference between the Pack approach and the algorithms of the Krimp family is that 0's and 1's are symmetrically considered in Pack.

The Stijl algorithm [TV12b] is a tree-based MDL approach taking into account both 0's and 1's and storing itemsets in a tree. However, contrasting Pack, Stijl relies on "tiles", i.e., rectangles in a dataset. The tree in Stijl is a hierarchy of nested tiles, where parent-child relations are inclusion relations between tiles. A child tile is created whenever its density –the relative number of 1's– differs a lot from the parent one. An extension of tile discovery is proposed in [FvL20] where "geometric pattern mining" with the Vouw algorithm is introduced. This algorithm considers arbitrarily shaped patterns in raster-based data, i.e., data tables with a fixed order of rows and columns, and can identify descriptive pattern sets even in noisy data. Finally, the discovery of tiles is also closely related to Boolean Matrix Factorization (BMF). In a nutshell, the objective of BMF is to find an approximation of a binary matrix C by a Boolean product of two low-rank matrices A and B. The columns of A and the rows of B describe the factors, which correspond to tiles. The MDL principle has also been applied to BMF to select the rank of the decomposition [MV14, MT20].

All the MDL-based algorithms which are surveyed above apply to binary or categorical data. Now we focus on a few algorithms which are dealing with pattern mining in numerical data. First of all the RealKrimp algorithm [WDKG14] is an extension of Krimp to real-valued data, where patterns are

axis-aligned hyper-rectangles. Even if the algorithm does not require any preprocessing, it actually needs a discretization of the data. Moreover, there is also a "feature selection" step where unimportant hyper-rectangle dimensions are removed. RealKrimp is tailored to mine high-density patterns, and to minimize the combinatorial explosion, it constructs each hyper-rectangle using a pair of neighboring rows sampled from the original dataset. Then, without prior knowledge about the data, the choice of the size of sampling is difficult as a too small sample size may output very general patterns, while a too large sample size may increase the execution time. The problem of an inappropriate sample size may be partially solved by setting a large "perseverance", i.e., how many close rows should be checked to improve compression when enlarging the hyper-rectangle, and "thoroughness", i.e., how many consecutive compressible patterns are tolerated. As it can be understood, finding optimal parameters in RealKrimp constitutes an important problem in the pattern mining process. Moreover, the hyper-rectangles in RealKrimp are evaluated independently, meaning that the algorithm searches for a set of optimal patterns instead of an optimal pattern set. The subsequent pattern redundancy may be mitigated by sampling data and computing the hyper-rectangles in different parts of the attribute space. Thereby, RealKrimp relies on many heuristics and has no means to jointly evaluate the set of generated hyper-rectangles. Besides, heuristics imply some prior knowledge about the data which is not always available in practice. All these aspects should be taken into account.

In [MKN19a] we proposed another approach to mine informative hyper-rectangles in numerical data. The approach can be summarized in 3 steps: (i) greedily computing dense hyper-rectangles by merging the closest neighbors and ranking them by prioritizing dense regions, (ii) greedily optimizing an MDL-like objective to select the intervals –sides of hyper-intervals– for each attribute independently, (iii) constructing the patterns using the selected intervals and maximizing the number of instances described by the intervals by applying a closure operator (a closed set is maximal for given support). This approach tends to optimize entropy, which is proportional to the length of data encoded by the intervals and does not take into account the complexity of the global model, i.e., the set of patterns. This simplification is based on the observation that each newly added interval replaces at least one existing interval, and thus, the complexity of the model does not increase. Moreover, the compression process is lossy as the data values can be reconstructed only up to some selected intervals. Finally, the approach allows for feature selection but does not address explicitly the problem of overlapping patterns.

Based on this first experience, below we propose Mint algorithm, which is based on MDL and aimed at mining patterns in numerical data. We restrict the patterns to be hyper-rectangles as they are the most interpretable types of multidimensional patterns. As in RealKrimp, the Mint algorithm deals with discretized data which allow us to define a lossless compression scheme. The problem of feature selection is not addressed while patterns may overlap. Finally, contrasting RealKrimp, Mint is less dependent on heuristics and discovers an approximation of an MDL-optimal pattern set rather than an approximation of single optimal patterns.

5.3 Basic notions

5.3.1 Formalization of data and patterns

Let D^* be a numerical dataset that consists of a set of objects $G = \{g_1, \ldots, g_n\}$ and a set of attributes $M = \{m_1, \ldots, m_k\}$. The number of objects and attributes is equal to n and k, respectively. Each attribute $m_i \in M$ is numerical and its range of values is denoted by $range(m_i)$. Each object $g \in G$ is *described* by a tuple of attribute values $\delta(g) = \langle v_i \rangle_{i \in \{1,\ldots,k\}}$.

As patterns we use axis-aligned hyper-rectangles, or "boxes". In multidimensional data, an axis-aligned hyper-rectangle has one of the simplest descriptions –a tuple of intervals– and thus can be easily analyzed by humans. The hyper-rectangle describing a set of objects B is given by

$$h = \langle [\min\{v_i \mid v_i \in \delta(g), g \in B\}, \max\{v_i \mid v_i \in \delta(g), g \in B\}] \rangle_{i \in \{1,\ldots,k\}} .$$

We call the i-th interval of a hyper-rectangle the i-th *side* of the hyper-rectangle. The support of a hyper-rectangle h is the number of objects that lie within the hyper-rectangle h, i.e., $sup(h) = |\{g \in G \mid \delta(g) \in h\}|$.

m_1	m_2
0.30	0.15
0.05	3.90
0.20	4.35
4.40	0.00
4.30	3.70
4.10	3.90
4.25	6.60
7.10	4.15
6.70	6.50
6.90	7.40
7.45	6.75
7.10	7.35

m_1	m_2	
0	0	h_1
0	4	h_2
0	4	
4	0	h_3
4	4	h_4
4	4	
4	7	h_5
7	4	h_6
7	7	h_7
7	7	
7	7	
7	7	

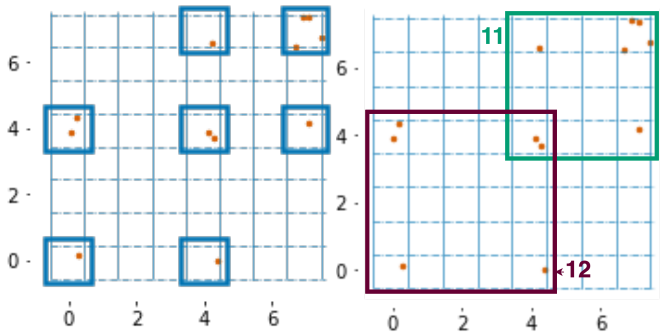


Figure 5.1: Dataset over attributes $\{m_1, m_2\}$ (left), its discretized version (middle), representation of the dataset on the plane and its partition into 8×8 equal-width intervals (right). The discretization grid is given by dotted lines, the corresponding non-empty elementary hyper-rectangles are given by dashed lines. The axis labels show the indices of elementary hyper-rectangles

Often, instead of continuous numerical data, one deals with discretized data, where the continuous range of values $range(m_i)$ is replaced by a set of integers, which are the indices of the intervals. Formally speaking, a range $range(m_i)$ is associated with a partition based on a set of intervals $\mathcal{B}_i = \{B_i^j = [c_{j-1}, c_j) \mid j = 1, \ldots, l\}$, where c_0 and c_l are the minimum and maximum values, respectively, of $range(m_i)$. Thus, each $v \in [c_{j-1}, c_j)$ is replaced by j.

The endpoints of the intervals can be chosen according to one of the methods considered above, e.g., equal-width, equal-height intervals, or using the MDL principle, and the number of the intervals may vary from one attribute range to another attribute range. The endpoints define a *discretization grid*. The number of grid dimensions is equal to the number of attributes.

A chosen discretization splits the space $\prod_{i=1}^{|M|} range(m_i)$ into a finite number of elementary hyper-rectangles $h_e \in \{\prod_{i=1}^{|M|} B_i^j \mid B_i^j \in \mathcal{B}_i\}$, i.e., each side of an elementary hyper-rectangle is composed of one discretization interval B_i^j. Non-elementary hyper-rectangles are composed of consecutive elementary hyper-rectangles.

For a hyper-rectangle $h = \left\langle [c_i^{(l)}, c_i^{(u)}) \right\rangle_{i \in \{1,\ldots,|M|\}}$, where $c_i^{(l)}$ and $c_i^{(u)}$ are endpoints of intervals from $\mathcal{B}_i$, in the discretized attribute space we define the *size* of the ith side as the number of elementary hyper-rectangles included into this side, i.e., $size(h, i) = |\{B_i^j \mid B_i^j \subseteq [c_i^{(l)}, c_i^{(u)})\}|$. Further, we use h to denote a hyper-rectangle (pattern), $\mathcal{H}$ to denote a set of hyper-rectangles (patterns), and D to denote the dataset D^* discretized w.r.t. the chosen discretization grid.

Example. Let us consider a dataset given in Fig. 5.1 (left). It consists of 12 objects described by attributes m_1 and m_2. All the descriptions are distinct (unique). Each attribute range is split into 8 intervals of width 1. The discretized dataset is given in Fig. 5.1 (right). It has 7 unique rows. The non-empty elementary hyper-rectangles correspond to non-empty squares induced by the 8×8 discretization grid. The number of hyper-rectangles is equal to the number of distinct rows in the discretized dataset (given in the middle).

5.3.2 Information theory and MDL

MDL [Grü07] is a general principle that is widely used for model selection and works under the slogan "the best model compresses data the best". As it was discussed in Section 2.3, the basic idea is to replace some regularities in data with the code words. The code words are associated in such a way that the most frequent regularities have the shortest code words. In this chapter we consider numerical data and, as regularities, we consider patterns (hyper-rectangles).

Formally speaking, given a dataset D the goal is to select such a subset of patterns $\mathcal{H}$ that minimizes the description length $L(D, \mathcal{H})$. In the crude version of MDL [Grü07] the description length is given by $L(D, \mathcal{H}) = L(\mathcal{H}) + L(D|\mathcal{H})$, where $L(\mathcal{H})$ is the description length of the model (set of patterns) $\mathcal{H}$, in bits, and $L(D|\mathcal{H})$ is the description length of the dataset D encoded with this set of patterns, in bits.

The length $L(\mathcal{H})$ characterizes the complexity of the set of patterns and penalizes high-cardinality pattern sets, while the length of data $L(D|\mathcal{H})$ characterizes the conformity of patterns w.r.t. the data. $L(D|\mathcal{H})$ increases when the patterns are too general and do not conform well with the data. Thus, taking into account both $L(\mathcal{H})$ and $L(D|\mathcal{H})$ allows achieving a balance between the pattern set complexity and its conformity with the data.

Roughly speaking, the minimization of the total length consists in (i) designing the encoding scheme, (ii) choosing patterns that are specific for a given dataset, and (iii) assigning to these patterns the code words allowing for a shorter total length $L(D, \mathcal{H})$.

In MDL, our concern is the length of the code words rather than the codes themselves. That is why we use a real-valued length instead of an integer-valued length.

Intuitively, the length of code words is optimal when shorter code words are assigned to more frequently used patterns. From the information theory, given a probability distribution over $\mathcal{H}$, the length of the Shannon prefix code for $h \in \mathcal{H}$ is given by $l(h) = -\log P(h)$ and is optimal. We write log for $\log_2$. Then we obtain the following probability model: given the usage $usg(h)$ of $h \in \mathcal{H}$ in the encoding. The probability distribution that ensures an optimal pattern code length for the chosen encoding scheme is given by

$$P(h) = \frac{usg(h)}{\sum_{h_i \in \mathcal{H}} usg(h_i)},$$

where the usage $usg(h)$ of a pattern h is the number of times a pattern h is used to cover objects G in a dataset D. However, this model is based on the assumption that the total number of encoded instances (the length of the transmitted sequence) is known. Moreover, to encode/decode the message, the transmitter should know the usages $usg(h)$ of all patterns $h \in \mathcal{H}$ and the receiver should know the corresponding probability distribution, it needs to be transmitted, this is not usually the case.

Prequential plug-in codes [Grü07] do not have this kind of limitation. These codes refer to "online" codes since they can be used to encode sequences of arbitrary lengths, and they do not require to know in advance the usage of each code word. The codes are based on only previously encoded instances. Moreover, they are asymptotically optimal even without any prior knowledge on the probabilities [Grü07]. The prequential plug-in codes are widely used in recent MDL-based models [FvL20, PvL20, BV15].

More formally, the idea of the prequential codes is to assess the probability of observing the n-th element h^n of the sequence based on the previous elements $h^1, \ldots, h^{n-1}$. Thus prequential codes allow for a predictive-sequential interpretation for arbitrary length sequences.

Let H^n be the sequence $h^1, \ldots, h^{n-1}, h^n$. The probability of the n-th pattern $h^n \in \mathcal{H}$ in the pattern sequence H^n is given by

$$P_{plug\text{-}in}(h^n) = \frac{\prod_{h \in \mathcal{H}} [\Gamma(usg(h) + \varepsilon)/\Gamma(\varepsilon)]}{\Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}, \tag{5.1}$$

where $usg(h)$ is the number of occurrences of pattern h in the sequence H^n, and $usg(\mathcal{H}) = \sum_{h \in \mathcal{H}} usg(h)$ is the length of the sequence, i.e., the total number of occurrences of patterns from $\mathcal{H}$, ε is a pseudocount, i.e., the usage of a pattern when it has no counts, and $\Gamma(x) = \int_0^1 (-\log(t))^{x-1}\, dt$ is the gamma function. We give the technical details of the derivation of Equation 5.1 in Section 5.3.3.

Then the length of the code word associated with h^n is given as follows:

$$\begin{aligned} l(h^n) &= -\log P_{plug\text{-}in}(h^n) = \\ &= \log \Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|) - \log \Gamma(\varepsilon|\mathcal{H}|) - \sum_{h \in \mathcal{H}} \left[\log \Gamma(usg(h) + \varepsilon) - \log \Gamma(\varepsilon)\right]. \end{aligned} \tag{5.2}$$

As was mentioned above, we are interested in the *length* of the code words rather than in the code words themselves. That is why we use real-valued length instead of integer-valued length for the number of bits needed to store the real code words.

To encode integers, when it is needed, we use the *standard universal code for the integers* [Ris83] given by $L_{\mathbb{N}}(n) = \log n + \log\log n + \log\log\log n + \ldots + \log c_0$, where the summation stops at the first negative term, and $c_0 \approx 2.87$ [Grü07].

5.3.3 Derivation of the plug-in codes

Let H^n be the sequence $h^1, \ldots, h^{n-1}, h^n$. The idea of the prequential codes is to assess the probability of observing the n-th element h^n of the sequence H^n based on the previous elements $h^1, \ldots, h^{n-1}$: $P_{plug\text{-}in}(h^n) = \prod_{i=1}^{n} P(h^i|h^{i-1})$.

Initially, a uniform distribution over $\mathcal{H}$ is defined with a pseudo-count ε over the set of patterns $\mathcal{H}$, i.e., the probability of h^0 is given by $P(h^0) = \frac{\varepsilon}{\varepsilon \cdot |\mathcal{H}|}$. Then, during the process of transmitting/receiving messages, the pattern probabilities and lengths are updated w.r.t. the patterns observed so far.

At each single step, the distribution P over $\mathcal{H}$ is multinomial with parameters $(\theta_1, \ldots, \theta_{|\mathcal{H}|})$, where θ_i corresponds to a pattern $h_i \in \mathcal{H}$. With the subscript indices we arbitrarily enumerate patterns in $\mathcal{H}$. The superscript indices denote the sequence number of patterns in the transmitting sequence. Further, we will see that the order of patterns in the sequence $h^1, \ldots, h^n$ does not affect the length of the encoded sequence. This length depends only on the number of times each pattern appears in the sequence.

Taking into account the initial probabilities, the maximum-likelihood estimates of the parameters of the multinomial distribution, given the sequence $H^n = h^1 \ldots h^n$, are the following:

$$\hat{\theta}_{h^n}(h) = \frac{usg(h|h^1 \ldots h^n) + \varepsilon}{\sum_{h^* \in \mathcal{H}} usg(h^*|h^1 \ldots h^n) + \varepsilon|\mathcal{H}|}, \tag{5.3}$$

where $usg(h|h^1 \ldots h^n)$ is the number of occurrences of pattern h in the sequences observed so far (up to the n-th pattern inclusive). The ML estimates from Equation 5.3 are equivalent to the probability estimates of a pattern h based on its frequency in H^h with Laplace smoothing having parameter ε [MRS08].

Thus, taking as the probability model the multinomial distribution with the parameters estimated according to the maximum likelihood principle, the plug-in probability of the n-th pattern in the sequence H^n is given by

$$P_{plug\text{-}in}(h^n) = \prod_{i=1}^{n} P(h^i|h^{i-1}) = \prod_{i=1}^{n} P_{\hat{\theta}_{h^{i-1}}}(h^i). \tag{5.4}$$

Combining together Equations 5.3 and 5.4 we obtain the following probability of the n-th pattern $h^n \in \mathcal{H}$ in the pattern sequence H^n:

$$P_{plug\text{-}in}(h^n) = \frac{\prod_{h \in \mathcal{H}} \prod_{j=0}^{usg(h)-1}(j+\varepsilon)}{\prod_{j=0}^{usg(\mathcal{H})-1}(j+\varepsilon|\mathcal{H}|)} = \frac{\prod_{h \in \mathcal{H}} \Gamma(usg(h)+\varepsilon)/\Gamma(\varepsilon)}{\Gamma(usg(\mathcal{H})+\varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}, \tag{5.5}$$

where $usg(h)$ is the number of patterns in H^n, and $usg(\mathcal{H}) = \sum_{h \in \mathcal{H}} usg(h)$ is the length of the sequence, i.e., the total number of occurrences of patterns from $\mathcal{H}$. Then, the code length of h^n is given as follows:

$$\begin{aligned} l(h^n) = -\log P_{plug\text{-}in}(h^n) &= \sum_{i=1}^{n} -\log P_{\hat{\theta}_{H^{i-1}}}(h^i) = \\ &= \log\Gamma(usg(\mathcal{H})+\varepsilon|\mathcal{H}|) - \log\Gamma(\varepsilon|\mathcal{H}|) - \sum_{h \in \mathcal{H}} \left[\log\Gamma(usg(h)+\varepsilon) - \log\Gamma(\varepsilon)\right]. \end{aligned}$$

The length $l(h^n)$ can be interpreted as the sum of the log loss of the prediction errors made by sequentially predicting h^i based on the predictor in the family of multinomial distributions over $\mathcal{H}$ that would have been the best for sequentially predicting the previous patterns $h^1, \ldots, h^{i-1}$.

5.3.4 ML-estimates of the parameters of the multinomial distribution

Let $\mathcal{H}$ be a set of n patterns, i.e., $n = |\mathcal{H}|$ and x_i be the number of times $h_i \in \mathcal{H}$ appears in the sequence of patterns of length N. Then, the probability mass function of the multinomial distribution for the given

sequence has the following form:

$$p(x_1, \ldots, x_n \mid p_1, \ldots, p_n) = \frac{N!}{x_1! \ldots x_n!} \prod_{i=1}^{n} p_i^{x_i}.$$

The log-likelihood function then is the following:

$$\ell(p_1, \ldots, p_n) = \log(N!) - \sum_{i=1}^{n} \log(x_i!) + \sum_{i=1}^{n} x_i \log(p_i).$$

To find the ML estimates we need to maximize this function w.r.t. $p_1, \ldots, p_n$ given the constraint on the probabilities $\sum_{i=1}^{n} i = 1$. Applying the methods of Lagrange multipliers we get the following equation:

$$\mathcal{L}(p_1, \ldots, p_n, \lambda) = \ell(p_1 \ldots, p_n) + \lambda(1 - \sum_{i=1}^{n}(p_i)).$$

Thus,

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial p_i} = \frac{x_i}{p_i} - \lambda = 0, i = 1, \ldots, n \\ \frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_{i=1}^{n} p_i = 0. \end{cases}$$

Taking into account the pseudo-counts, we get $x_i = usg(h_i) + \varepsilon$, where $usg(h_i)$ is the number of times the pattern h_i appears in the sequence.

Thus,

$$\begin{cases} p_i = \frac{usg(h_i)+\varepsilon}{\lambda}, i = 1, \ldots, n \\ \sum_{i=1}^{n} p_i = 1. \end{cases}$$

Putting the last equation into the first one we get $p_i = \frac{usg(h_i)+\varepsilon}{\sum_{i=1}^{n} usg(h_i)+\varepsilon|\mathcal{H}|}$.

5.4 MINT

We propose an approach to pattern mining in multidimensional numerical discretized data. The main assumption we rely on is that all the attributes are important, i.e., patterns are computed in the whole attribute space. To apply this method we consider a discretized attribute space, i.e., each attribute range is split into equal-width intervals, as it was done in [WDKG14, NMVB14]. The choice of the equal-width intervals is due to the fact that the cost, in bits, of the reconstruction of a real value here is constant for all intervals. Each object therefore is included into an $|M|$-dimensional *elementary hyper-rectangle*. Starting from the elementary hyper-rectangles (each side is composed of one interval), we greedily generalize the currently best patterns and select those that provide the maximal reduction of the total description length. At each step we reuse some of the previously discovered candidates as well as other candidates computed on the fly using the last added pattern.

5.4.1 The model encoding

Firstly, we define the total description length of the set of hyper-rectangles and the data encoded by them. The total description length is given by $L(D, \mathcal{H}) = L(\mathcal{H}) + L(D \mid \mathcal{H})$, where $L(\mathcal{H})$ is the description length, in bits, of the set of hyper-rectangles $\mathcal{H}$, and $L(D|\mathcal{H})$ is the description length, in bits, of the discretized dataset encoded by this set of hyper-rectangles. The initial set of the hyper-rectangles is composed exclusively of elementary ones.

To encode the set of hyper-rectangles $\mathcal{H}$, we need to encode the discretization grid and the positions of the hyper-rectangles in this grid. Thus, the total length of the pattern set is given by

$$L(\mathcal{H}) = \underbrace{L_{\mathbb{N}}(|M|) + \sum_{i=1}^{|M|} L_{\mathbb{N}}(|\mathcal{B}_i|)}_{code\ length\ of\ the\ grid} + \underbrace{L_{\mathbb{N}}(|\mathcal{H}|) + |\mathcal{H}| \left(\sum_{i=1}^{|M|} \log\left(|\mathcal{B}_i|(|\mathcal{B}_i| + 1)/2\right) \right)}_{code\ length\ of\ the\ pattern\ set}.$$

To encode the grid we need to encode the number of dimensions (attributes) $|M|$ and the number of intervals $|\mathcal{B}_i|$ within each dimension i. This grid is fixed and is not changed throughout the pattern mining process. To encode the pattern set $\mathcal{H}$, given the grid, we need to encode the number of patterns $|\mathcal{H}|$ and the positions of their boundaries within each dimension. Since there exist $\binom{|\mathcal{B}_i|}{2} + |\mathcal{B}_i| = 0.5(|\mathcal{B}_i|(|\mathcal{B}_i|+1))$ possible positions of the boundaries within the i-th dimension, namely $\binom{|\mathcal{B}_i|}{2}$ combinations where the boundaries are different, and $|\mathcal{B}_i|$ cases where the lower and upper boundaries belong to the same interval, meaning only one interval from the grid is involved. These positions are encoded uniformly. The latter gives the cost of $\log(0.5|\mathcal{B}_i|(|\mathcal{B}_i|+1))$ bits for encoding the i-th side of a pattern within the chosen grid.

The code length of a dataset encoded with the set of patterns is given by

$$L(D|\mathcal{H}) = \underbrace{L_{\mathbb{N}}(|G|)}_{\substack{cost\ of\ encoding\ the \\ number\ of\ instances}} + \underbrace{L(D(\mathcal{H})|\mathcal{H})}_{\substack{cost\ of\ encoding \\ D\ with\ \mathcal{H}}} + \underbrace{L(D \ominus D(\mathcal{H})|\mathcal{H})}_{reconstruction\ cost}$$

where the first component encodes the number of objects, the second one corresponds to the length of data encoded with hyper-rectangles, and the third one corresponds to the cost of the reconstruction of the object description $\delta(g) = \langle v_i \rangle_{i \in \{1,\ldots,|M|\}}$ up to elementary intervals. Let us consider the last two components in detail.

The cost of the reconstruction of the true real values is constant for all values due to the equal-width discretization and it is not changed during pattern mining, thus is not taken into account in $L(D, \mathcal{H})$. The dataset is encoded by exploring all objects in a given order and assigning to each object a code word of the pattern covering this object. According to the MDL principle, each data fragment should be covered (encoded) only once, otherwise, the encoding is redundant. However, some patterns may overlap, i.e., include the same object. A *covering strategy* then defines which data fragment is an occurrence of which pattern.

Here, the usage is defined as follows: $usg(h) = |cover(h, G)|$. We discuss the covering strategy in detail in Section 5.4.2.

From Equation 5.2, the length of data encoded with the plug-in codes is given by

$$L(D(\mathcal{H})|\mathcal{H}) = \log \frac{\Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}{\prod_{h \in \mathcal{H}} \Gamma(usg(h) + \varepsilon)/\Gamma(\varepsilon)} =$$
$$\log \Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|) - \log \Gamma(\varepsilon|\mathcal{H}|) - \sum_{h \in \mathcal{H}} \left[\log \Gamma(usg(h) + \varepsilon) - \log \Gamma(\varepsilon)\right],$$

where $usg(\mathcal{H}) = \sum_{h \in \mathcal{H}} usg(h)$.

Once each object has been associated with a particular pattern, its original description within the pattern up to elementary intervals is encoded in $L(D \ominus D(\mathcal{H})|\mathcal{H})$. We use $D \ominus D(\mathcal{H})$ to denote the difference ("distortion") between the initially discretized dataset D and the same dataset encoded with $\mathcal{H}$.

To reconstruct the dataset up to the elementary equal-width intervals we encode the positions of each object within the corresponding pattern, this cost is given by

$$L(D \ominus D(\mathcal{H})|\mathcal{H}) = \sum_{h \in \mathcal{H}} usg(h) \log(size(h)) = \sum_{h \in \mathcal{H}} usg(h) \left(\sum_{i=1}^{|M|} \log(size(h, i)) \right),$$

where $size(h, i)$ is the number of elementary intervals that compose the side i of the pattern h.

Example. Let us consider an encoding of the data by patterns according to the model introduced above for the case of the running example. We take the set of two hyper-rectangles $\mathcal{H} = \{h_{11}, h_{12}\}$, which are given in Fig. 5.1. Let the cover of h_{11} be $cover(h_{11}, G) = \{g_5, \ldots, g_{12}\}$ and cover of h_{12} be $cover(h_{12}, G) = \{g_1, g_2, g_3, g_4\}$. Then, the encoding of the pattern set is given by $L(\mathcal{H}) = L_{\mathbb{N}}(2) + 2 \cdot L_{\mathbb{N}}(8) + L_{\mathbb{N}}(2) + 2 \cdot 2 \cdot \log 36$. Here, we need $2 \log 36$ bits to encode each pattern, i.e., $\log 36$ bits to encode uniformly 36 possible positions of the boundaries for each side of the hyper-rectangle.

The length of data encoded by $\mathcal{H}$ is given by $L(D(\mathcal{H})|\mathcal{H}) = L_{\mathbb{N}}(12) + \log \Gamma(12 + \varepsilon \cdot 2) - \log \Gamma(\varepsilon \cdot 2) - [\log \Gamma(8 + \varepsilon) - \log \Gamma(\varepsilon) + \log \Gamma(4 + \varepsilon) - \log \Gamma(\varepsilon)]$. The reconstruction error is equal to $L(D \ominus D(\mathcal{H})|\mathcal{H}) =$

$8 \cdot (\log(4) + \log(4)) + 4 \cdot (\log(5) + \log(5))$, i.e., we need to encode the positions of the data points within the corresponding hyper-rectangles up to the elementary hyper-rectangles.

As we can see from the example above, the patterns can overlap. In such a case, one relies on a covering strategy to decide which pattern to use to encode each object. In the next section we introduce the algorithm that defines this strategy and allows for computing patterns minimizing the total description length.

5.4.2 The MINT algorithm

The objective of the MINT algorithm is to compute in a numerical dataset a pattern set which is the best w.r.t. the MDL principle.

Computing Minimal Pattern Set. Let M be a set of continuous attributes, G be a set of objects having a description based on attributes M, $\mathcal{P}$ be a set of all possible $|M|$-dimensional hyper-rectangles defined in the space $\prod_{m \in M} range(m)$, and *cover* be a covering strategy. One main problem is to find the smallest set of hyper-rectangles $\mathcal{H} \subseteq \mathcal{P}$ such that the total compressed length $L(D, \mathcal{H})$ is minimal.

The pattern search space in numerical data, where patterns are hyper-rectangles, is infinite. Even considering a restricted space, where all possible boundaries of the hyper-rectangles are limited to the coordinates of the objects from G, the search space is still exponentially large. The introduced total length $L(D, \mathcal{H})$ does not exhibit (anti)monotonicity property over the pattern set and thus does not allow us to exploit some efficient approaches to its minimization. Hence, to minimize $L(D, \mathcal{H})$, we resort to heuristics.

Main algorithm

Accordingly, the main idea of MINT is the following. Starting from elementary hyper-rectangles, we sequentially discover patterns that minimize the description length $L(D, \mathcal{H})$ by merging a pair of currently optimal patterns from $\mathcal{H}$. To compute a candidate pattern based on a pair of other patterns, we introduce the *join* operator $\oplus$. For two hyper-rectangles $h_j = \langle [v_i^{(l)}, v_i^{(u)}) \rangle_{i \in \{1,\ldots,|M|\}}$ and $h_k = \langle [w_i^{(l)}, w_i^{(u)}) \rangle_{i \in \{1,\ldots,|M|\}}$, their join is given by the smallest hyper-rectangle containing them, i.e., $h_j \oplus h_k = \langle [\min(v_i^{(l)}, w_i^{(l)}), \max(v_i^{(u)}, w_i^{(u)})) \rangle_{i \in \{1,\ldots,|M|\}}$.

We determine the *cover* of $h_j \oplus h_k$ as the union of the covers of h_j and h_k, i.e., $cover(h_j \oplus h_k, G) = cover(h_j, G) \cup cover(h_k, G)$. Thus, the usage of $h_j \oplus h_k$ is simply the cardinality of its cover, i.e., $usg(h_j \oplus h_k) = |cover(h_j \oplus h_k, G)|$. Since the elementary hyper-rectangles form a partition on G, it follows from the definition of *cover* that the usage is additive, i.e., $usg(h_j \oplus h_k) = usg(h_j) + usg(h_k)$. Thus, each object is covered by a single pattern from $\mathcal{H}$. We say that pattern h *encodes* an object if it *covers* it.

Among all candidates $\{h_j \oplus h_k \mid h_j, h_k \in \mathcal{H}\}$ we consider as the best ones those that ensure the largest gain in the total description length:

$$\begin{aligned}
&\Delta L(\mathcal{H}, D, h_j, h_k) = L(D, \mathcal{H}) - L(D, \mathcal{H} \cup \{h_j \oplus h_k\} \setminus \{h_j, h_k\}) = \\
&\underbrace{L_{\mathbb{N}}(|\mathcal{H}|) - L_{\mathbb{N}}(|\mathcal{H}| - 1) + \left(\sum_{i=1}^{|M|} \log\left(|\mathcal{B}_i|(|\mathcal{B}_i| + 1)/2\right) \right)}_{\Delta L(\mathcal{H})} + \\
&\underbrace{\log \frac{\Gamma(|G| + \varepsilon|\mathcal{H}|)}{\Gamma(|G| + \varepsilon(|\mathcal{H}| - 1))} + \log \frac{\Gamma(\varepsilon(|\mathcal{H}| - 1))}{\Gamma(\varepsilon|\mathcal{H}|)} - \log \frac{\Gamma(usg(h_j) + \varepsilon)\Gamma(usg(h_k) + \varepsilon)}{\Gamma(usg(h_j \oplus h_k) + \varepsilon)\Gamma(\varepsilon)}}_{\Delta L(D(\mathcal{H}))} + \\
&\underbrace{usg(h_j)size(h_j) + usg(h_k)size(h_k) - usg(h_j \oplus h_k)size(h_j \oplus h_k)}_{\Delta L(D \ominus D(\mathcal{H}))}.
\end{aligned} \tag{5.6}$$

The term "gain" stands for the difference between the total description lengths obtained using the current pattern set $\mathcal{H}$ and the pattern set where patterns h_j and h_k are replaced the its joined pattern $h_j \oplus h_k$. The gain is the largest for the candidates that compress better a given dataset.

Algorithm 6 MINT($D^*, |\mathcal{B}_i|$, k, N)

Input: numerical dataset D^*,
number of intervals for each attribute $|\mathcal{B}_i|$, $i = 1, \ldots, |M|$,
number of the nearest neighbors for computing the candidates k
number of candidates for pruning N
Output: pattern set $\mathcal{H}$,
total description length $\mathcal{L}_{total}$

1: $D \leftarrow DiscretizeData(D^*, \{|\mathcal{B}_i| \mid i = 1, \ldots, |M|\})$
2: $\mathcal{H} \leftarrow GetElementaryHyper\text{-}rectangles(D)$
3: $\mathcal{L}_{total} \leftarrow L(D, \mathcal{H})$
4: $\mathcal{C} \leftarrow \cup_{h_j \in \mathcal{H}, h_k \in NN(h_j,k)}(h_j \oplus h_k, \Delta L(\mathcal{H}, D, h_j, h_k))$
5: **while** $|\mathcal{C}| > 0$ **do**
6: $\quad \mathcal{C}_{new} \leftarrow \emptyset$
7: $\quad (h_j \oplus h_k, \Delta L) \leftarrow PopLargestGain(\mathcal{C})$
8: $\quad$ **while** $\Delta L > 0$ **do**
9: $\quad\quad$ **if** $h_j, h_k \in \mathcal{H}$ **then**
10: $\quad\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{h_j \oplus h_k\} \setminus \{h_j, h_k\}$
11: $\quad\quad\quad \mathcal{C}_{new} \leftarrow \mathcal{C}_{new} \cup \{(h_j \oplus h_k) \oplus h \mid h \in \mathcal{H}, h \neq h_j \oplus h_k\}$
12: $\quad\quad\quad \mathcal{L}_{total} \leftarrow \mathcal{L}_{total} - \Delta L$
13: $\quad\quad$ **end if**
14: $\quad\quad (h_j \oplus h_k, \Delta L) \leftarrow PopLargestGain(\mathcal{C})$
15: $\quad$ **end while**
16: $\quad \mathcal{C} \leftarrow \mathcal{C}_{new}$
17: $\quad$ MINT-PRUNING($D, \mathcal{H}, N$)
18: **end while**
19: **return** $\mathcal{H}$, $\mathcal{L}_{total}$

The minimization of $L(D|\mathcal{H})$ consists in an iterative process where candidates are computed using pairs of currently optimal patterns and selecting one that provides the largest gain in the total description length.

The pseudocode of MINT is given in Algorithm 6. At the beginning, the optimal patterns $\mathcal{H}$ are the elementary hyper-rectangles (line 2) induced by an equal-width discretization into a chosen number of intervals $|\mathcal{B}_i|$, $i = 1, \ldots, |M|$. We also set an additional parameter k to limit the number of candidates by k nearest neighbors. For large datasets and large number of intervals $|\mathcal{B}_i|$, setting a low value $k \ll |G|$ reduces the computational efforts. In the discretized space, the elementary hyper-rectangles are points, thus the distance between them is the Euclidean distance. Then, given a hyper-rectangle $h_j \oplus h_k$, the neighbors are the neighbors of h_j and h_k. The main loop in lines 5-18 consists in selecting the best candidates from the current set of candidates $\mathcal{C}$, updating the set of optimal patterns $\mathcal{H}$, and collecting new candidates in $\mathcal{C}_{new}$. Once all candidates from $\mathcal{C}$ have been considered in the inner loop (lines 8-15), the new candidates from $\mathcal{C}_{new}$ become current ones (line 16). In the inner loop, lines 8-15, the candidates that minimize the total description length are selected one by one. They are considered by decreasing gain ΔL. At each iteration of the inner loop, the candidate $h_j \oplus h_k$ providing the largest gain ΔL is taken, the corresponding optimal patterns h_j and h_k are replaced with $h_j \oplus h_k$ in $\mathcal{H}$. The candidates based on the newly added patterns are added to $\mathcal{C}_{new}$ (line 11) and are not considered at the current iteration. In $\mathcal{C}_{new}$ we store pairs of indices making new candidates, and only in line 16 we compute candidates by calculating the gains ΔL that they provide. These gains, however, are computed only for patterns $h_j \oplus h_k$ where both h_j, h_k are still present in the set $\mathcal{H}$. Postponing the computation of candidates to line 16 allows us to reduce the number of candidates and to speed up pattern mining. The outer loop stops when there are no more candidates in $\mathcal{C}$. After that, the pruning can be performed (line 17). We consider the details of the pruning strategy after the next example.

Example. Let us consider how the algorithm works on the running example from Fig. 5.1. Initially, the set of hyper-rectangles consists of elementary ones, i.e., $h_1, \ldots, h_7$ (Fig. 5.2). We restrict the set

pattern	coordi-	nates	a nearest neighbor
h_1	0	0	h_2
h_2	0	4	h_1
h_3	4	0	h_1
h_4	4	4	h_5
h_5	4	7	h_4
h_6	7	4	h_4
h_7	7	7	h_5

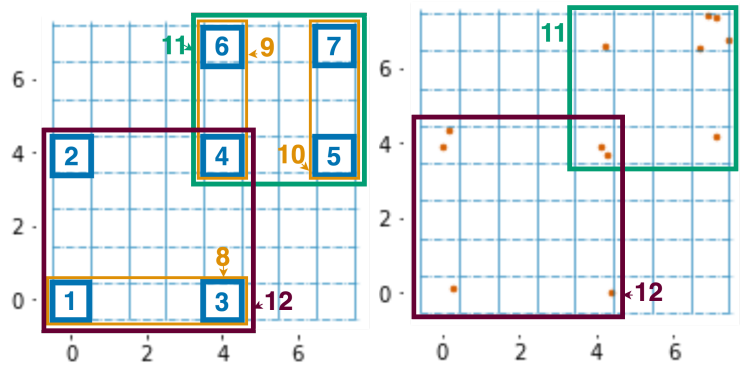


Figure 5.2: The sequential changes in the set of currently optimal patterns. The initial set of hyper-rectangles is composed of elementary ones, i.e., $h_1, \ldots, h_7$

of candidates by considering only one nearest neighbor for each pattern. If a pattern has more than one nearest neighbor, we select a pattern with the smallest index. The nearest neighbors for each elementary hyper-rectangle are given in Fig. 5.2 (left). Thus, the set of candidates is given by $\mathcal{C} = \{h_1 \oplus h_2, h_1 \oplus h_3, h_4 \oplus h_5, h_4 \oplus h_6, h_5 \oplus h_7\}$. The patterns are added in the following order: $h_8 = h_1 \oplus h_3$, $h_9 = h_4 \oplus h_6$, $h_{10} = h_5 \oplus h_7$, which corresponds to decreasing gain. After that, set $\mathcal{C}$ does not contain candidates that can improve the total length. Thus, MINT proceeds by considering the candidates from $\mathcal{C}_{new} = \{h_j \oplus h_k \mid j, k = 2, 8, 9, 10, j \neq k\}$, which are the candidates composed of pairs of recently added patterns and the unused old ones. The newly added patterns are $h_{11} = h_9 \oplus h_{10}$ and $h_{12} = h_2 \oplus h_8$. The new candidate set is $\mathcal{C}_{new} = \{h_{11} \oplus h_{12}\}$. The algorithm terminates, the set of hyper-rectangles corresponding to the smallest total description length is $\mathcal{H} = \{h_{11}, h_{12}\}$.

Complexity of Mint. At the beginning, the number of candidates, i.e., pairs of elementary hyper-rectangles (line 4) is $O(min(|\mathcal{H}|^2, k \cdot |\mathcal{H}|))$, where $|\mathcal{H}|$ is the number of non-empty elementary hyper-rectangles. The number of elementary hyper-rectangles does not exceed the number of objects $|G|$. Thus, setting $k \ll |\mathcal{H}|$ we have the number of candidates $O(k \cdot |\mathcal{H}|) \sim (k \cdot |G|)$ which is linear w.r.t. the number of objects. Further, the number of hyper-rectangles can only decrease. Computing a candidate $h_j \oplus h_k$ takes $O(|M|)$. The number of candidates added to $\mathcal{C}_{new}$ at each iteration of the inner loop is equal to $|\mathcal{H}| - 2$, $|\mathcal{H}| - 3$, etc. Thus, the total number of candidates is $O(|\mathcal{H}|^2)$. The total complexity of computing candidates is $O(|M| \cdot |\mathcal{H}|^2)$. Searching for the maximal gain among the candidates takes $O(|\mathcal{H}|^2)$ time. Since there can be at most $\mathcal{H}$ iterations, computing the maximal gain takes $O(|\mathcal{H}|^3)$. In the worst case, where $|\mathcal{H}| = |G|$, the complexity is $O(|M||G|^2 + |G|^3)$, however it is possible either to set a small number of initial intervals $|\mathcal{B}_i|$ that ensure $|\mathcal{H}| \ll |G|$ or to restrict the number of candidates.

The theoretical complexity of the considered algorithms is estimated based on different features. In REALKRIMP, the size of the sampling s mainly affects the time complexity, the cost of computing the first hyper-rectangle is $O(|M|s^2 \log s + |M||G|s)$, additional ones are mined in $O(|M|s^2 + |M||G|s)$. In the worst case, where the sample size is proportional to the number of objects, the time complexity is $O(|M||G| \log |G|)$ and $O(|M||G|^2)$ for the first and additional hyper-rectangles, respectively. In MINT the main component is the number of hyper-rectangles which is at most $|G|$, thus, the total time complexity is $O(|M||G|^2 + |G|^3)$. Thus, both REALKRIMP and SLIM have polynomial complexity w.r.t. the dataset size. SLIM has the largest worst-case complexity $O(|\mathcal{C}|^3|G||I|)$, where $|\mathcal{C}| = O(2^{\min(|G|,|I|)})$ is the number of the candidates, which can be exponential in the size of the dataset. Moreover, the size of the dataset used in SLIM is larger than the size of the dataset used by MINT and REALKRIMP, since the number of attributes $|I|$ in a binarized dataset is larger than the number of attributes $|M|$ in the discretized one. Thus, REALKRIMP and MINT have polynomial complexity in the input size. However, in practice, MINT works much faster as shown in the experiments.

Algorithm 7 MINT-PRUNING($D, \mathcal{H}, N$)

Input: dataset D,
pattern set $\mathcal{H}$,
the maximal number of considered candidates k
Output: pattern set $\mathcal{H}$

1: $H_{old} \leftarrow |\mathcal{H}| + 1$
2: **while** $H_{old} > |\mathcal{H}|$ **do**
3: $H_{old} \leftarrow |\mathcal{H}|$
4: $\mathcal{C} \leftarrow \{h_j \oplus h_k \mid h_j, h_k \in \mathcal{H}, \exists h \in \mathcal{H}, h \subseteq h_j \oplus h_k, h \neq h_j \neq h_k\}$
5: **for all** $h_j \oplus h_k \in GetTopNBy\Delta(\mathcal{C})$ **do**
6: $\mathcal{S} = \emptyset$
7: $\Delta \leftarrow \Delta L(\mathcal{H}, D, h_j, h_k)$
8: **for all** $h \in h_j \oplus h_k$, $h \in \mathcal{H}$ **do**
9: $cover(h_j \oplus h_k, G) = cover(h_j \oplus h_k, G) \cup cover(h, G)$
10: $\Delta_h \leftarrow \Delta L(\mathcal{H} \setminus (\mathcal{S} \cup \{h\}), D, h_j, h_k)$
11: **if** $\Delta_h > \Delta$ **then**
12: $\mathcal{S} \leftarrow \mathcal{S} \cup \{h\}$
13: $\Delta \leftarrow \Delta_h$
14: **else**
15: $cover(h_j \oplus h_k, G) = cover(h_j \oplus h_k, G) \setminus cover(h, G)$
16: **end if**
17: **end for**
18: **if** $\Delta > 0$ **then**
19: $\mathcal{H} = \mathcal{H} \setminus (\mathcal{S} \cup \{h_j, h_k\}) \cup \{h_j \oplus h_k\}$
20: **end if**
21: **end for**
22: **end while**
23: **return** $\mathcal{H}$

Pruning strategy

In some cases, it may happen that merging a pair of patterns does not ensure minimization of the total description length while merging several patterns at once does provide a shorter total description length. We propose a pruning strategy that consists in merging several candidates when the candidates generated by a pair of patterns do not provide a shorter description length. The pseudo-code of this procedure is given in Algorithm 7.

The outer loop in lines 2-22 is executed while there are some candidates allowing for a shorter total length, i.e., the number of patterns in $\mathcal{H}$ decreases. At each iteration of the outer loop, a new set of candidates $\mathcal{C}$ is created (line 4). As candidates we consider all pairs h_j, h_k such that the smallest hyper-rectangle including $h_j \oplus h_k$ also contains at least one pattern $h \in \mathcal{H}$ that differs from h_j and h_k. We order candidates by decreasing gain ΔL and consider at the current iteration the top-N candidates.

Then, in the inner loop in lines 5-21, we check candidates $h_j \oplus h_k$ one by one. In contrast to MINT, we do not require anymore that each candidate $h_j \oplus h_k$ improves the total description length since in lines 8-17 we extend it with other patterns. Those patterns that improve the gain are stored in $\mathcal{S}$. Once all hyper-rectangles contained in $h_j \oplus h_k$ have been considered, the set $\mathcal{S} \cup \{h_j \oplus h_k\}$ contains all patterns that will be replaced with $h_j \oplus h_k$, if this candidate ensures a positive gain in the total description length (lines 18-20).

Complexity of Pruning. In the beginning, the number of candidates is $O(|\mathcal{H}|^2)$, where $|\mathcal{H}|$ is the number of the currently optimal patterns. This value does not exceed the number of elementary hyper-rectangles that, in turn, is not greater than $|G|$. We may restrict the number of considered candidates $\mathcal{C}$ to N (line 5). Thus, setting $N \ll |\mathcal{H}|$ we have a linear number of candidates at each iteration, however, without restrictions, the number of candidates is $O(|\mathcal{H}|^2)$. Computing a candidate from a pair of patterns

takes $O(|M|)$. For each candidate we search for the included patterns, which takes $O(|M||\mathcal{H}|)$ time. Thus, in total we have $O(|\mathcal{H}|^2(|M| + |M||\mathcal{H}|) \sim O(|M||\mathcal{H}|^3)$ for one iteration of the inner loop. Since there can be at most $O(|\mathcal{H}|)$ iterations of the outer loop, the worse total complexity of pruning is $O(|M||\mathcal{H}|^4)$. However, $\mathcal{H}$ and the number of iterations of the outer loop are usually small, and pruning is performed fast. Otherwise, the set of candidates can be limited to the top N.

5.5 Experiments

In this section, we compare Mint with Slim and RealKrimp –the most similar MDL-based approaches to numerical pattern mining.

Slim and RealKrimp are not fully comparable with our approach since they have their own parameters that actually affect both the performance and the quality of the results. Slim works with binarized data, while Mint and RealKrimp work with discretized data. However, Slim and Mint allow for choosing the number of discretization intervals. Slim is better adapted to mine patterns in datasets with a coarse discretization. A coarse discretization results in a moderate increase in the number of attributes, while a fine discretization usually results in a drastic increase and can make the task intractable. By contrast, Mint is able to mine patterns efficiently even in datasets with fine discretization. A last difference is that Slim and Mint evaluate a pattern set as a whole, while RealKrimp evaluates each pattern in a set independently.

5.5.1 Datasets

We selected datasets from the UCI repository [?] that are generally used in itemset mining, and discretized with the LUCS-KDD DN software [Coe03]. Since Mint implicitly discretizes datasets into equal-width intervals, we were compelled to use the same data discretization for other algorithms. The choice of the optimal number of intervals depends both on the chosen algorithm and the dataset. In experiments we do not tune the parameters specifically for each dataset and each algorithm. We considered the parameters that work generally well for the considered algorithms, namely splitting into 5 intervals (as in LUCS-KDD repository, where the number of intervals is limited to 5 for each attribute) and into $\sqrt{|G|}$.

Since the proposed discretization into 5 intervals may be non-optimal, we also use IPD discretization, which is expected to provide only necessary discretization intervals, thus should be the most suitable for Slim. Moreover, as Slim deals with binary data, we additionally transform each discretization interval into a binary attribute. The parameters of the real-world datasets are given in Table 5.1.

Parameters of the methods. To compare the aforementioned methods we chose the following parameters. For Slim and Mint, that allow for choosing the discretization strategy, we chose the discretization into 5 and $\sqrt{|G|}$ equal-width intervals. The first one is expected to be more suitable for Slim, while the second one is expected to give better results for Mint. For Mint we additionally set the number of neighbors, considered for computing candidates, equal to the number of intervals. We also use discretization by interaction-preserving discretization (IPD) [NMVB14] — an unsupervised multivariate MDL-based discretizer — under the assumption that this method provides an optimal splitting of the attribute ranges. Since Slim deals with binary data, we transform each discretization interval into a binary attribute and use these data in Slim.

RealKrimp relies on a set of parameters, the most important among them are *sample size*, *perseverance*, and *thoroughness*. Sample size defines the size of the dataset sample that will be used to compute patterns. We consider samples of size $\sqrt{|G|}$, $0.25|G|$ and $0.5|G|$. The sample size affects the running time: the smaller samples allow for faster pattern mining operations, while too small samples may prevent to discover interesting patterns. Perseverance regulates the behavior of RealKrimp to reach a global minimum of the description length. Large values of perseverance help to reach a global minimum. Perseverance is then set to 1/5 of the sample size. Thoroughness is the maximal number of consecutive non-compressible hyper-intervals that should be considered. We set thoroughness equal to 100. The parameters of the real-world datasets are given in Table 5.1.

Table 5.1: Description of datasets and the parameters of the discretization (applicable to MINT and SLIM). All attributes are numerical

name	$\|G\|$	$\|M\|$	#intervals			classes
			5	$\sqrt{\|G\|}$	IPD	
iris	150	4	20	45	8	3
wine	178	13	65	163	26	3
haberman	306	3	15	42	6	2
ecoli	336	7	29	91	14	8
breast wisconsin	569	30	146	581	84	2
spambase	4601	57	266	1794	149	2
waveform	5000	40	200	2681	198	3
parkinsons	5875	18	87	961	92	-
statlog satimage	6435	36	180	2335	213	6
gas sensor	13910	128	566	8915	1060	27
avila	20867	10	37	445	88	12
credit card	30000	24	112	1401	195	2
shuttle	58000	8	40	459	91	77
sensorless dd	58509	48	219	5022	693	11
mini	130064	50	117	1623	1031	2
workloads	200000	5	25	2109	296107	-
average	**33425**	**30**	**132**	**1792**	**254**	-

Compression ratio. Firstly, we consider the main quality measure of the MDL-based methods, namely *compression ratio*. REALKRIMP mines each pattern independently from others, thus the compression ratio of a pattern set is not measurable in this case. SLIM and MINT have different encoding schemes and compress binary and discretized datasets, respectively. Thus the compression ratios are not fully comparable. However, for the sake of completeness, we report the compression ratios that they provide in Fig. 5.3. As it was expected, SLIM works better in the case of a coarse discretization. For example, for equal-width discretization into 5 intervals, SLIM provides much better compression than MINT for datasets such as "aliva", "shuttle", "sensorless dd", and "mini". However, MINT works consistently better for fine discretized dataset (Fig. 5.3 in the middle, where the number of intervals is $\sqrt{|G|}$). This can be explained by the fact that IPD and MINT compress the same dataset, while SLIM uses an additional binarization step, thus it compresses the dataset containing less information since in the binarized datasets the interval order is not preserved. Thus, the better compression of SLIM may be partially explained by artifacts caused by this data transformation.

Total description length of the compressed data. Another important characteristic of MDL-based approaches is the total length resulting from compression. Because of the different forms of data used by SLIM and MINT (binary and discrete, respectively), the algorithms may have quite different compressed total description lengths. Nevertheless, we compare them. The ratio of the total length of SLIM by the total length of MINT is given in Fig. 5.4. As in the case of the compression ratio, REALKRIMP does not allow for computing the total length of data encoded by a pattern set, it computes only length gains provided by single patterns. Fig. 5.4 shows that the total length of SLIM is usually about 2 times greater than the total length of MINT. The latter can be explained, in particular, by the redundancy caused by data binarization. Then we may conclude that the encoding of the discretized data provided by MINT is more concise than the encoding of the binarized data used in SLIM.

Running time. The next important question is the running time, which is reported in Table 5.2. The cases, where pattern sets are not computed, are filled in the table with "...". The performance of SLIM and MINT is affected by the number of discretization intervals, while the performance of REALKRIMP

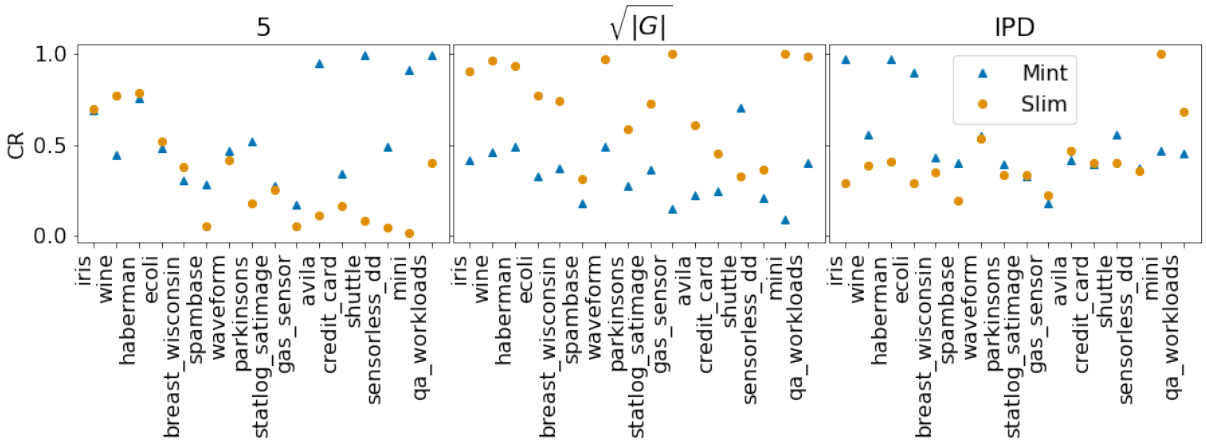


Figure 5.3: Compression ratios of SLIM and MINT for different discretizations (into 5 intervals, $\sqrt{|G|}$ and by IPD discretizer), the smaller values are better

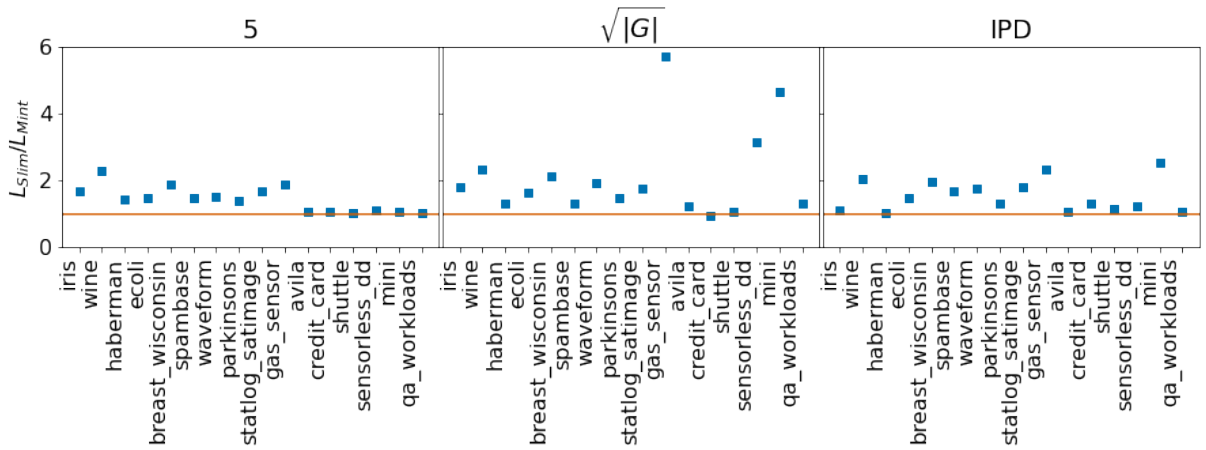


Figure 5.4: The ratio of the total description length of SLIM by the total description length MINT. The values close to 1 (horizontal line) corresponds to the cases where the total description lengths provided by both methods are almost the same

depends heavily on the sample size.

The running time reported in the table shows that for small datasets ($< 1k$) all methods work fast and often complete the work in less than a second. For average-sized ($< 50k$) and large ($> 50k$) datasets the running time depends on the chosen parameters. SLIM mines fast patterns in datasets with a coarse discretization (into 5 intervals). However, for fine discretizations (into $\sqrt{|G|}$ or IPD) the running time increases drastically. For example, for "shuttle" dataset, SLIM terminates in 1 second, when the number of intervals is 5, while to mine patterns when the attribute ranges are split into $\sqrt{|G|}$ intervals and by IPD, SLIM requires 17394 and 650 seconds, respectively. Again for SLIM, the scalability issues are especially pronounced for datasets with a large number of attributes, e.g., for "gas sensor" dataset, SLIM terminates in 8206, ...[11], and 72488 seconds, while MINT requires only 43, 1088, and 1000 seconds for the same discretization parameters.

REALKRIMP also suffers from poor scalability: for average- or large-sized datasets, setting a small sample size, e.g., $\sqrt{|G|}$, does not allow to find a sufficient amount of interesting patterns, while setting a reasonable sample size ($0.25|G|$ or $0.5|G|$) results in a drastic increase of the running time and memory

[11]The experiments are not completed within one week.

Table 5.2: Running time, in seconds, of Slim and Mint and RealKrimp

name	$\|G\|$	$\|M\|$	Slim			Mint			RealKrimp		
			# intervals			# intervals			sample size		
			5	$\sqrt{\|G\|}$	IPD	5	$\sqrt{\|G\|}$	IPD	$\sqrt{\|G\|}$	$0.25\|G\|$	$0.5\|G\|$
iris	150	4	0	0	0	0	0	0	0	0	0
wine	178	13	1	0	0	0	0	0	0	0	0
haberman	306	3	0	0	0	0	0	0	0	0	0
ecoli	336	7	0	0	0	0	0	0	0	1	1
breast w.	569	30	3	53	3	2	1	3	0	7	22
spambase	4601	57	11	6159	63	6	88	117	6	14169	18133
waveform	5000	40	505	50022	1346	5064	194	23	2	5510	11940
parkinsons	5875	18	3	599	27	12	129	85	6	999	1515
statlog s.	6435	36	790	32212	1876	330	266	219	8	2849	7140
gas sensor	13910	128	8206	...[a]	72488	43	1088	1000	113	42677	136802
avila	20867	10	1	4395	79	5	5796	1135	69	8854	28855
credit card	30000	24	38	29422	17083	611	15856	1900	171	192057	410780
shuttle	58000	8	1	17394	650	0	138	191	200	...[b]	...[b]
sensorless	58509	48	60	500263	83819	52	137213	11910	508	299237	...[b]
mini	130064	50	22	...[a]	...[a]	2	56006	18292	861	...[b]	...[b]
workloads	200000	5	27	113195	8177	14	128941	6305	1031	...[b]	...[b]

[a] Interrupted process. Experiments are not completed after one week.

[b] Interrupted process because of the lack of memory (requires more than 64GB).

requirement.

Our experiments show that Mint, dealing with the same discretization as Slim, requires less time to mine patterns, especially for large datasets. However, it is not enough to assess the performance of the patterns by considering their running time. It is also important to study how many patterns they return and which kind of patterns.

Number of MDL-selected patterns. Intuitively, the number of patterns should be small but sufficiently enough to describe all interesting relations between attributes. The number of MDL-selected patterns for the studied methods is reported in Table 5.3. The table shows that, given the same discretization, Slim returns usually a larger number of patterns than Mint. As in the case of the running time, Slim is sensitive to a large number of attributes and in this case usually returns a much larger number of patterns than Mint. For example, for "gas sensor" dataset Slim returns 1608, ...[12], and 9554 patterns, while Mint, with the same discretization settings, returns only 216, 566 and 773 patterns, respectively.

RealKrimp, on the contrary, returns a much smaller number of patterns than Mint and Slim. For example, for "gas sensor" dataset it returns only 4, 30, and 49 patterns for the samples of size $\sqrt{|G|}$, $0.25|G|$, and $0.5|G|$, respectively. Taking into account the running time, we can conclude that with the chosen parameters, the average running time per pattern is much larger for RealKrimp than for Slim and Mint. Thus, RealKrimp has the highest "cost" in seconds of generating a pattern. Now, let us examine the quality of the generated patterns.

Pattern similarity (redundancy). This parameter is particularly important for RealKrimp, where patterns are mined w.r.t. other patterns, but evaluated independently, and there are no guarantees for avoiding the selection of very similar patterns. To study pattern similarity, we consider the average

[12] The experiments are not completed within one week.

Table 5.3: The number of MDL-selected patterns by SLIM and MINT for discretization into 5, $\sqrt{|G|}$ intervals and the IPD discretization

name	$\|G\|$	$\|M\|$	SLIM # intervals 5	SLIM # intervals $\sqrt{\|G\|}$	SLIM # intervals IPD	MINT # intervals 5	MINT # intervals $\sqrt{\|G\|}$	MINT # intervals IPD	REALKRIMP sample size $\sqrt{\|G\|}$	REALKRIMP sample size $0.25\|G\|$	REALKRIMP sample size $0.5\|G\|$
iris	150	4	19	18	9	9	9	6	2	4	6
wine	178	13	85	62	42	15	11	17	1	4	9
haberman	306	3	13	12	8	6	9	3	2	0	0
ecoli	336	7	50	47	22	14	10	12	2	7	12
breast w.	569	30	246	564	219	37	33	64	19	12	23
spambase	4601	57	301	1486	881	78	201	445	2	95	97
waveform	5000	40	1726	4071	1645	345	140	30	1	65	199
parkinsons	5875	18	256	1418	708	82	207	377	4	21	25
statlog s.	6435	36	1322	6480	1871	345	260	395	3	32	41
gas sensor	13910	128	1608	...	9554	216	566	773	4	30	49
avila	20867	10	135	2149	1187	50	622	1385	8	23	33
credit card	30000	24	733	4002	3757	697	1225	1917	9	115	189
shuttle	58000	8	57	2947	1530	21	962	1623	5	0	0
sensorless	58509	48	571	14040	10466	404	1299	2257	4	27	0
mini	130064	50	150	...	...	39	901	7100	2	0	0
workloads	200000	5	688	4773	5889	297	3238	4592	8	0	0

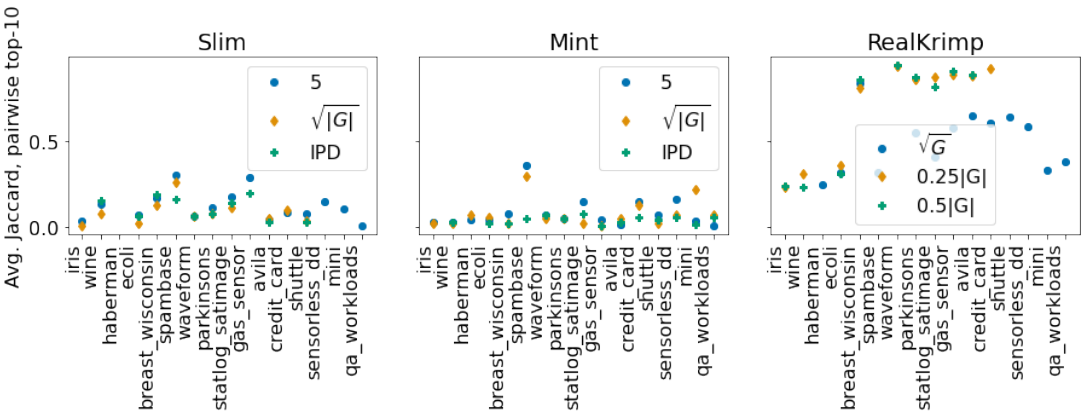


Figure 5.5: The average Jaccard similarity computed on the top-10 Jaccard-similar pairs for each pattern, repetitive pairs are discarded

pairwise Jaccard similarity computed w.r.t. the sets of objects that the patterns describe. We take into account all occurrences of patterns in data rather than their usage in the data covering (which is, by definition, non-redundant).

However, the average pairwise Jaccard similarity is sensitive to the number of patterns in the set, i.e., for large pattern sets having several groups of very similar patterns, Jaccard similarity may be quite low and does not spot this "local" redundancy.

To tackle this issue, we do not consider the pairwise similarity between all pairs but rather the similarity between the most similar pairs, i.e. for each pattern we select at most 10 patterns among the most similar patterns w.r.t. Jaccard similarity. Then we compute the average value of similarity by removing the repetitive pairs of patterns if there are any. The average values of similarity are presented in Fig. 5.5.

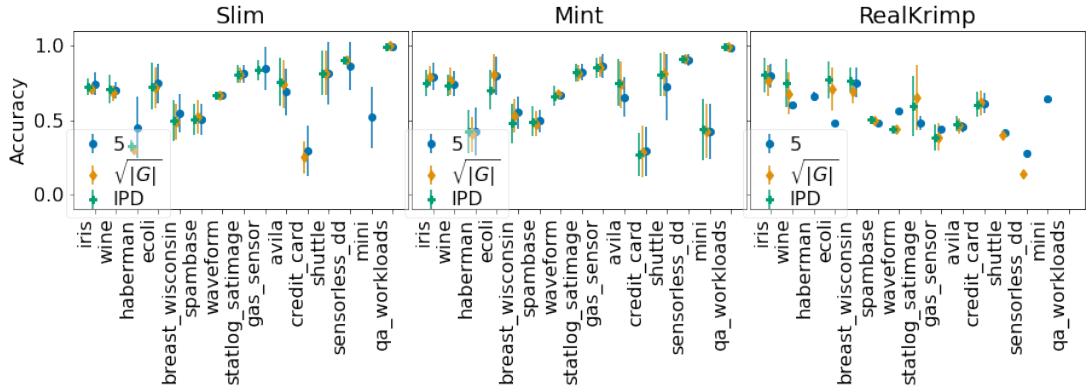


Figure 5.6: The average accuracy of patterns

The results of the experiments show that on average the pairwise Jaccard similarity is the smallest for Mint, and only slightly higher for Slim. In Slim higher values are caused by the fact that each object can be covered by different non-overlapping itemsets, thus these increased values of the Jaccard similarity are partially caused by the specificity of the model. RealKrimp has the largest values of the Jaccard similarity, close to 1 (see Fig. 5.5). This result is quite expected since the patterns are evaluated independently, thus the method does not minimize redundancy in the pattern set.

Purity of patterns. To evaluate the meaningfulness of the resulting patterns, we measure their accuracy by considering the classes of objects they describe. The class labels are not used during pattern mining and are considered only for assessing the pattern quality.

In Fig. 5.6, the results show that Slim and Mint, being based on the same discretizations, have quite similar average accuracy. RealKrimp return patterns with high accuracy for small datasets, however, loses in accuracy on large datasets.

Accuracy of pattern descriptions. As was mentioned in the introduction, in pattern mining it is important not only to describe meaningful groups of objects but also to provide quite precise boundaries of these groups. Unfortunately, we cannot evaluate how precise are the pattern boundaries for the real-world datasets, since we do not have any ground truth.

To evaluate how precise the boundaries of patterns we use synthetic datasets. We generate 6 types of 2-dimensional datasets with different numbers of patterns and different positions of patterns w.r.t. other patterns. The generated types of datasets are shown in Fig. 5.7. The ground truth patterns are highlighted in different colors. Further, we use $\mathcal{T}$ to denote a set of ground truth hyper-rectangles. For all these types of data, we generate datasets where each pattern contains 100, 200, 500, 700, 1000 objects. In Fig. 5.7, the "simple" datasets consist of separable patterns. The "variations" datasets contain adjacent patterns and thus allow for variations in pattern boundaries.

The "inverted" datasets include the most complicated patterns for Mint and Slim since they treat asymmetrically dense and sparse regions. It means that these algorithms are not able to identify the hole in the middle. Instead of this hole, we may expect a complicated description of the dense region around this hole. "Simple overlaps" contains overlapping patterns, while "simple inclusion" and "complex inclusion" can also contain patterns that are subsets of other patterns.

For all studied heuristics, namely discretization into 5, $\sqrt{|G|}$ equal-width intervals, and an IPD-discretization, Slim returns non-empty elementary rectangles induced by the chosen discretization, i.e., it does not merge any rectangles induced by the discretization grid. Thus, the quality of Slim-generated patterns is completely defined by the chosen discretization.

For Mint we consider the same discretization settings as for Slim, namely discretization into 5, $\sqrt{|G|}$ equal-width intervals, and an IPD-discretization. We study the different settings of Mint in the next section. For RealKrimp we study only default settings: samples of size $0.5|G|$. For the discussion of the

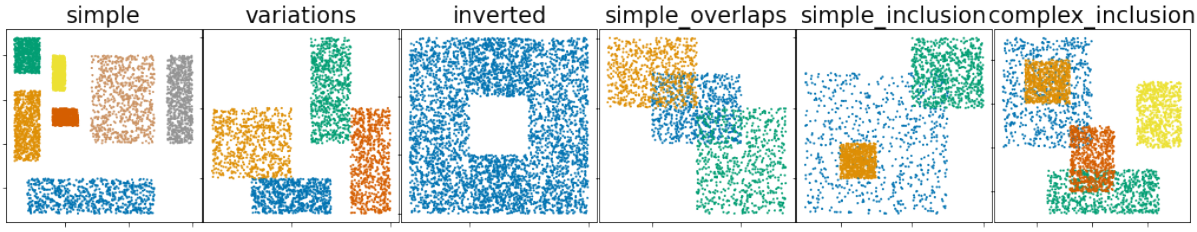


Figure 5.7: Six types of the generated synthetic dataset

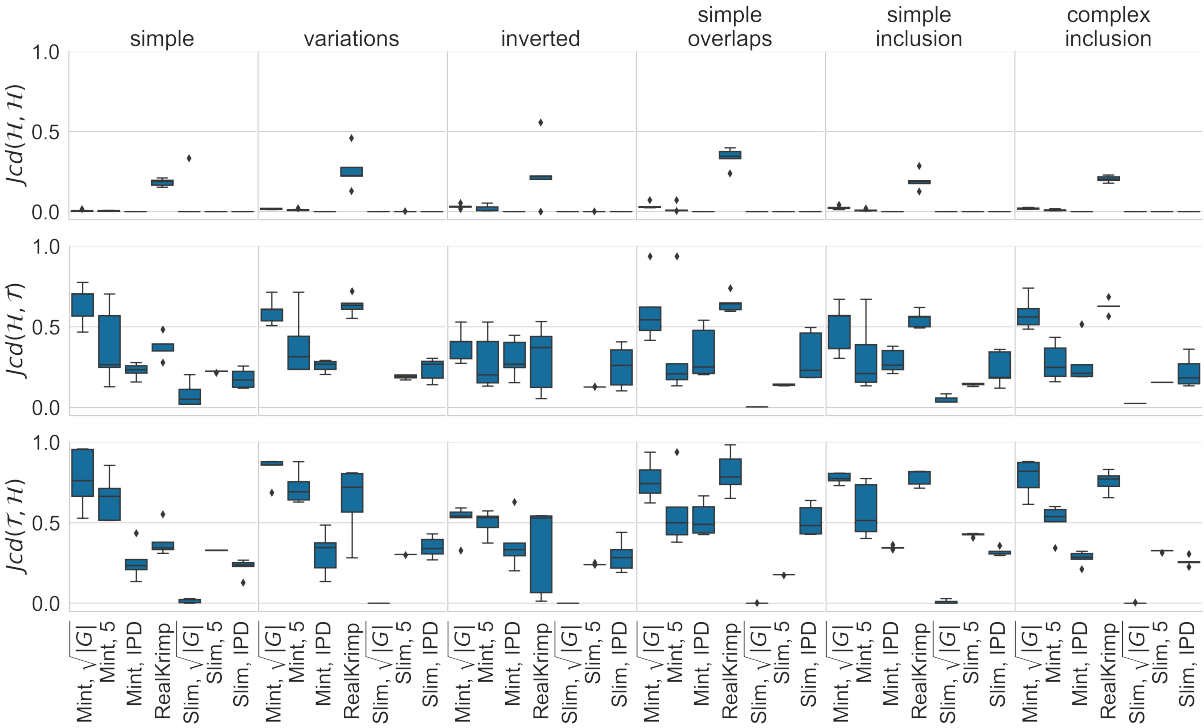


Figure 5.8: The average Jaccard similarity of hyper-rectangles

default settings of RealKrimp see [Wit12].

We evaluate the quality of patterns using the Jaccard similarity applied to hyper-rectangles. For two hyper-rectangles h_1 and h_2 the Jaccard similarity is given by

$$Jaccard(h_1, h_2) = area(h_1 \cap h_2)/area(h_1 \oplus h_2),$$

where $h_1 \cap h_2$ and $h_1 \oplus h_2$ is the intersection and join of h_1 and h_2, respectively.

We begin with the average pairwise Jaccard similarity of the computed patterns. The values reported in Fig. 5.8 (top row) show that Slim returns non-redundant patterns since they are non-overlapping, while RealKrimp returns very similar patterns. These patterns are redundant since even for the "simple" datasets, where all ground truth patterns are separable and non-overlapping, the patterns returned by RealKrimp are very similar. The similarity of Mint-selected patterns is very low, but it increases for the datasets with overlapping patterns, e.g., "simple overlaps" or "simple inclusion", and is almost 0 for the datasets with non-overlapping patterns, e.g., "simple" or "variations".

The next question is how well the boundaries of the computed patterns from $\mathcal{H}$ are aligned with the boundaries of the ground truth patterns from $\mathcal{T}$, i.e., the patterns that we generated. We evaluate them by the average Jaccard similarity between the computed patterns and the most similar ground truth

patterns as follows:

$$Jcd(\mathcal{H}, \mathcal{T}) = \frac{\sum_{h_1 \in \mathcal{H}} \max_{h_2 \in \mathcal{T}} Jaccard(h_1, h_2)}{|\mathcal{H}|}$$

.

In Fig. 5.8 (second row) we show the average values of $Jcd(\mathcal{H}, \mathcal{T})$. The values close to 1 indicate that all the computed patterns are very similar to the patterns given in ground truth. The worst results corresponds to Slim: even with a "smart" IPD discretization, the boundaries computed by Slim are not very precise. The low values for fine discretized data are explained by the inability of Slim to merge elementary hyper-rectangles.

Mint with the IPD-discretization returns also quite poor results. However, in the best settings (discretization into $\sqrt{|G|}$ intervals), Mint and RealKrimp have quite high values of $Jcd(\mathcal{H}, \mathcal{T})$. The latter means that all patterns from $\mathcal{H}$ are quite similar to the patterns given by ground truth.

Let us study the best patterns from each set, i.e., instead of considering all patterns we evaluate the quality of the best of them. We consider the average Jaccard similarity between the ground-truth patterns and the most similar patterns among the computed ones using the following formula:

$$Jcd(\mathcal{T}, \mathcal{H}) = \frac{\sum_{h_1 \in \mathcal{T}} \max_{h_2 \in \mathcal{H}} Jaccard(h_1, h_2)}{|\mathcal{T}|}.$$

The results are reported in Fig. 5.8 (third row). The values close to 1 correspond to the case where each ground truth pattern has at least one pattern in $\mathcal{H}$ that is very similar to ground truth patterns. The results given in Fig. 5.8 shows that the quality of Mint-generated patterns for the default discretization into $\sqrt{|G|}$ intervals is the best. For some datasets RealKrimp works equally well, e.g., "simple overlaps" or "simple inclusion", but for others it may provide quite bad results, e.g., for the simplest set of patterns contained in the "simple" datasets.

Comparing the values of $Jcd(\mathcal{H}, \mathcal{T})$ and $Jcd(\mathcal{T}, \mathcal{H})$ we may conclude that RealKrimp contains a lot of similar patterns, but these patterns do not match the ground truth patterns as well as the patterns generated by Mint. Thus, the experiments show that Mint (with the fine discretization) returns patterns with quite precise boundaries and outperforms the state-of-the-art MDL-based pattern miners.

Visualization of some hyper-rectangles for synthetic data. In this section we give some examples of the hyper-rectangles discovered by Slim, Mint, and RealKrimp. As was mentioned above, uniform boundaries used for an itemset miner such as Slim) may provide poor results, i.e., the boundaries of the hyper-rectangles can be very imprecise. And this is especially the case for the IPD discretization. Actually, IPD returns MDL-selected boundaries which are selected more for the overall distribution of points rather than for specific patterns. The results of IPD are given in Fig. 5.9.

The figure shows that IPD, especially for sparse data, provides quite imprecise boundaries. But even for dense data, the boundaries are not very well aligned to the ground truth patterns. Then we cannot expect good results for Slim and Mint when they are applied to IPD-discretized data.

Let us consider the patterns computed by Slim and Mint based on IPD-discretized data. Our experiments show that for the synthetic data Slim and Mint return quite similar patterns (see Fig. 5.10 and 5.11). We do not consider the IPD-discretized datasets further, since for the remaining datasets the anticipated results can be derived from the discretization grid given in

Due to limited space, we show patterns for randomly chosen datasets in Fig. 5.12-5.16. The ground truth patterns are highlighted in different colors in Fig. 5.7. As we can see Slim returns usually very imprecise patterns. This is a typical limitation of itemset mining methods applied to numerical data.

RealKrimp returns patterns with much more precise boundaries, however, a lot of patterns are almost the same. For example, for "simple" dataset, Fig. 5.12, some patterns generated by RealKrimp are very redundant and imprecise. The patterns returned by Mint are much more precise. They are non-redundant, however, there are patterns with imprecise boundaries (the 5th and the last one). The 1st pattern describes only a fragment of the ground truth pattern.

For dataset "simple inclusion", Mint returns two patterns that correspond exactly to the ground truth patterns and the remaining three patterns describe the third pattern. Nevertheless, their union gives a quite correct pattern. RealKrimp also distinguishes only 2 patterns correctly. However, it returns a lot

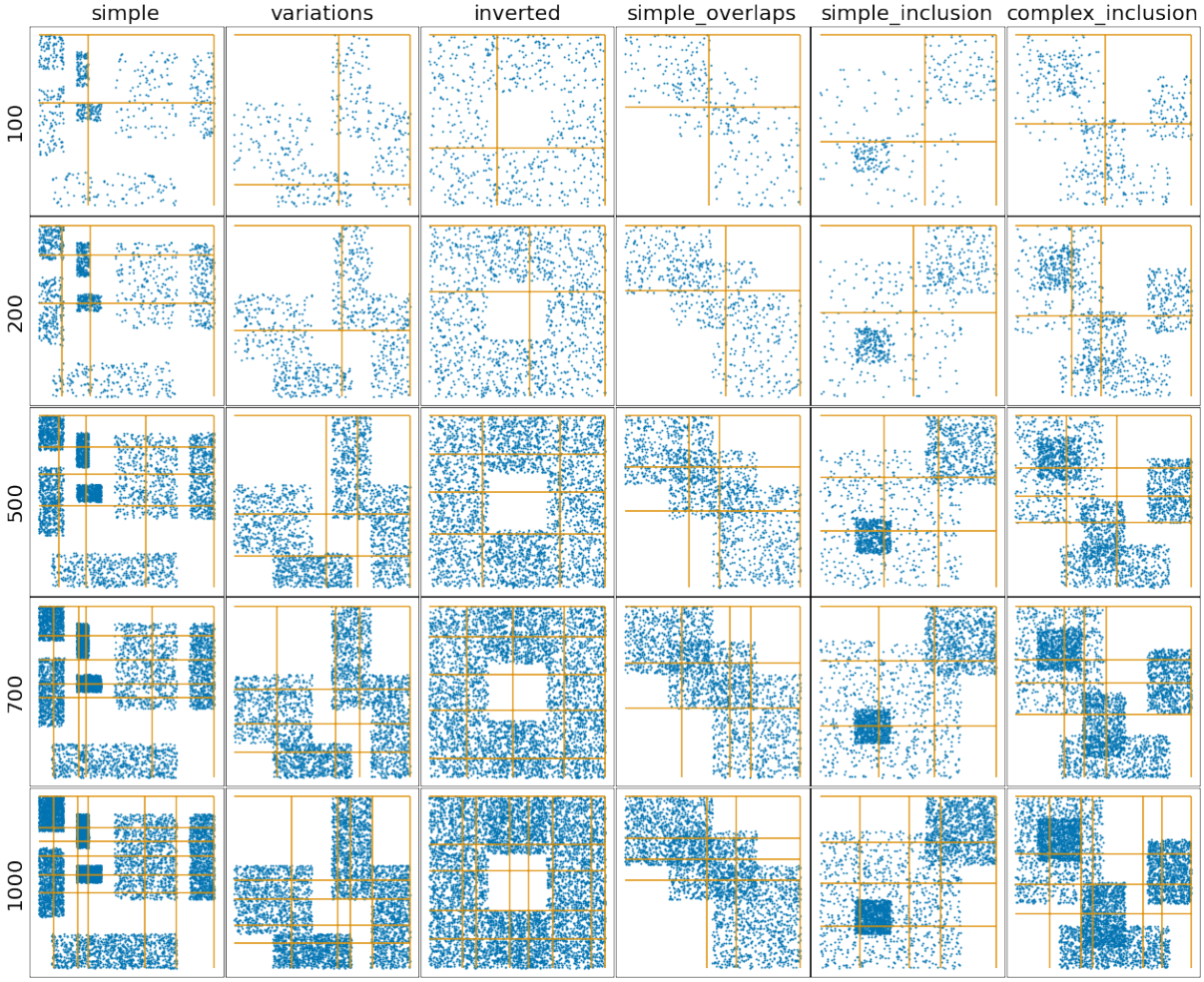


Figure 5.9: The results of IPD discretization for the synthetic data

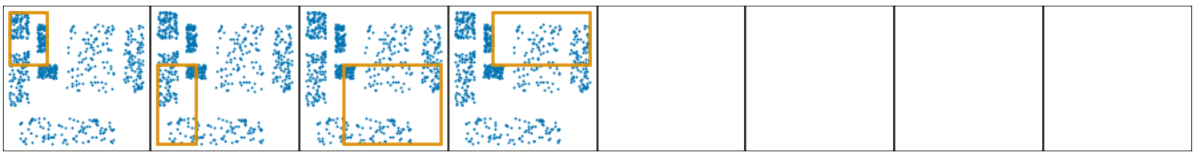


Figure 5.10: The patterns selected by MINT and SLIM (they are the same) for "simple" dataset, where support of each ground truth pattern is equal to 100, discretized by IPD

of similar patterns. It is important to notice that for this pattern set it is hard to find a combination of patterns that allows for reconstructing the third pattern. Thus, in the case of REALKRIMP, redundant patterns pose a greater problem than in the case of MINT.

The conclusion similar to the previous one, can be drawn for Fig. 5.14 and 5.15. Both MINT and REALKRIMP return patterns aligned to the ground truth patterns, however, the pattern sets returned by MINT are much less redundant, while REALKRIMP returns very similar patterns such that without knowing the ground truth it might be hard to choose those that are the closest to the correct ones.

The last case that we consider is given in Fig. 5.16. It contains a sparse pattern that MINT and SLIM are unable to describe, while REALKRIMP can easily do that. As we can see from the figure, instead of a sparse hyper-rectangle, MINT returns a cover of the dense region around this pattern. It is the

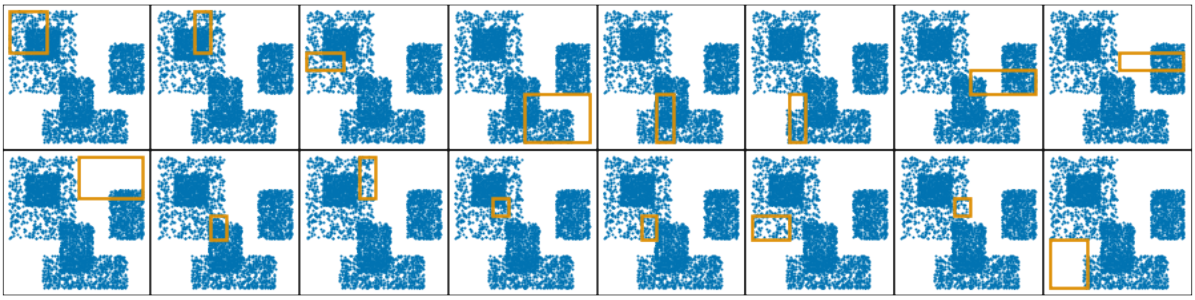
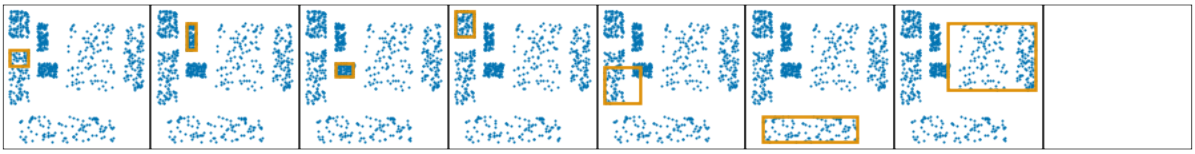
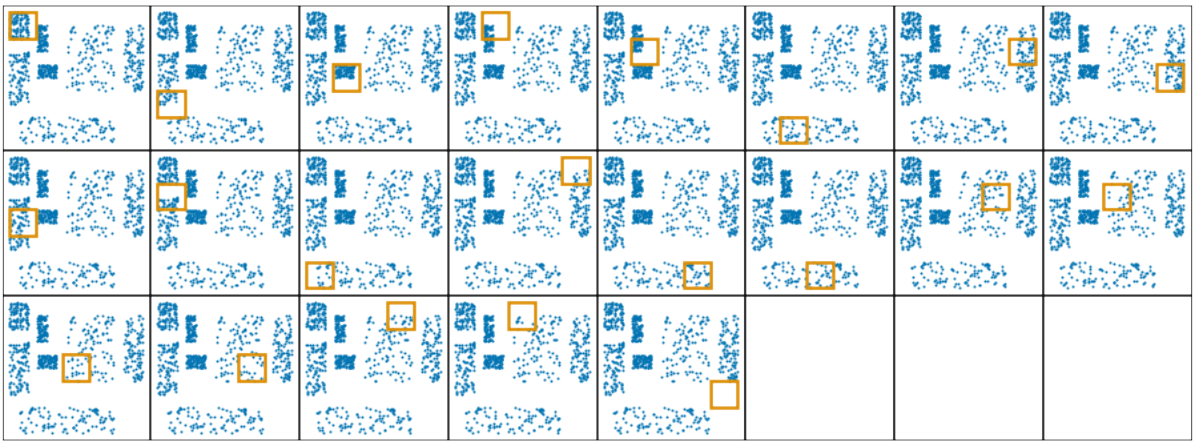


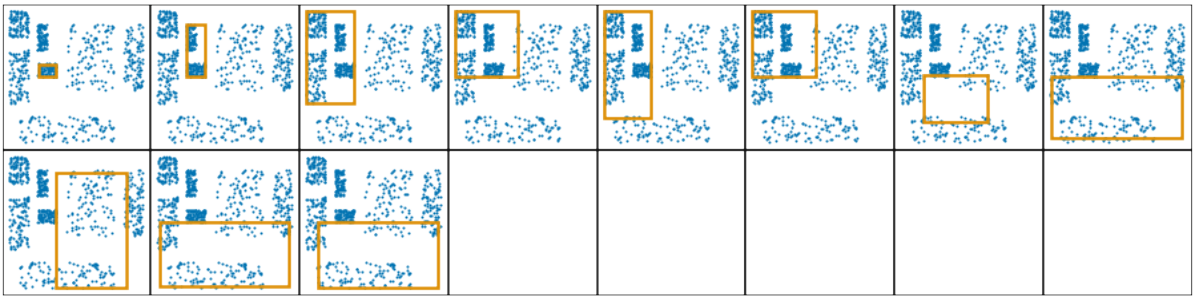
Figure 5.11: The patterns selected by MINT and SLIM (they are the same) for "complex inclusion" dataset, where support of each ground truth pattern is equal to 700, discretized by IPD



(a) MINT ($\sqrt{|G|}$)

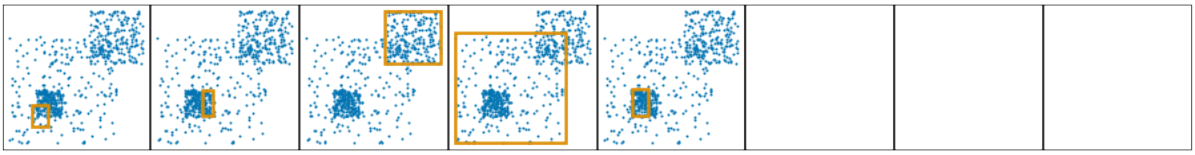


(b) SLIM (5)

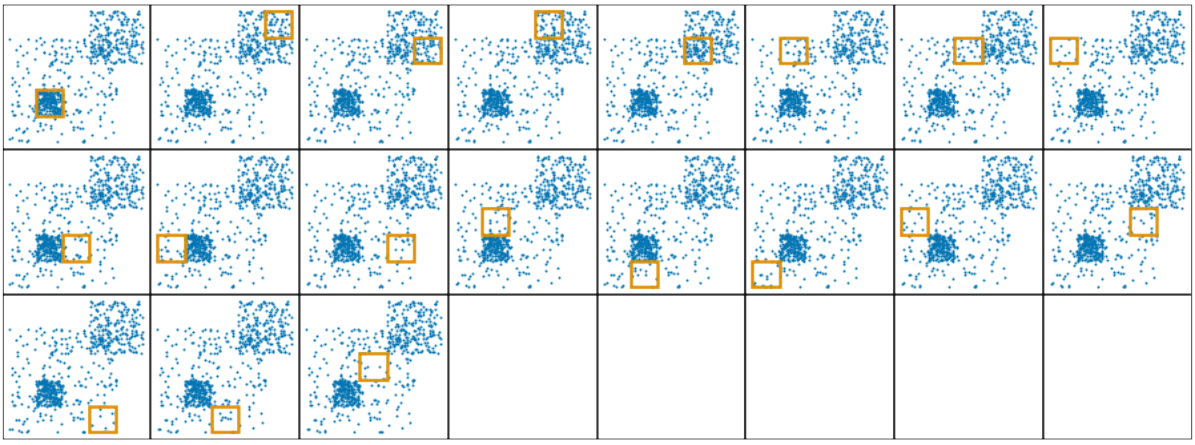


(c) REALKRIMP (sampling of size $0.5|G|$)

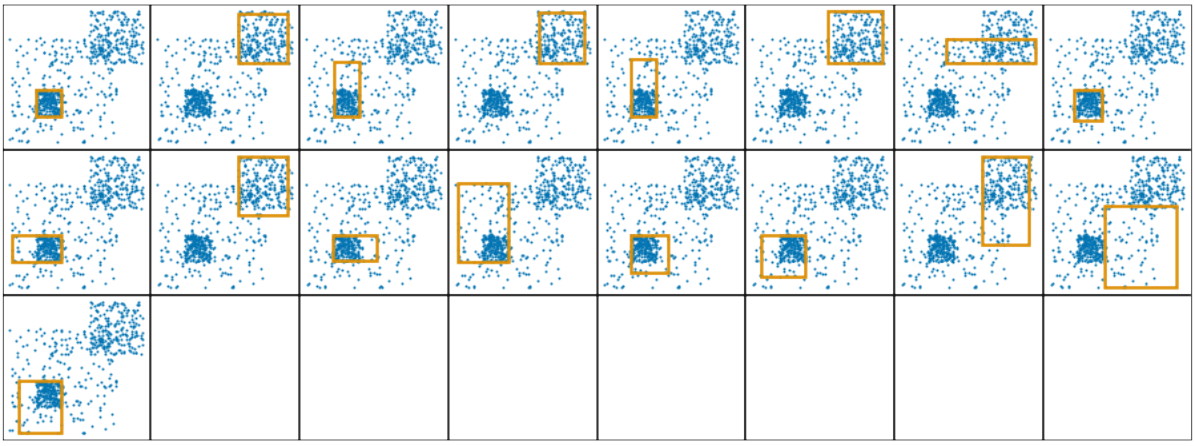
Figure 5.12: The results of pattern mining for "Simple" dataset, support of the ground truth patterns is 100



(a) MINT ($\sqrt{|G|}$)



(b) SLIM (5)



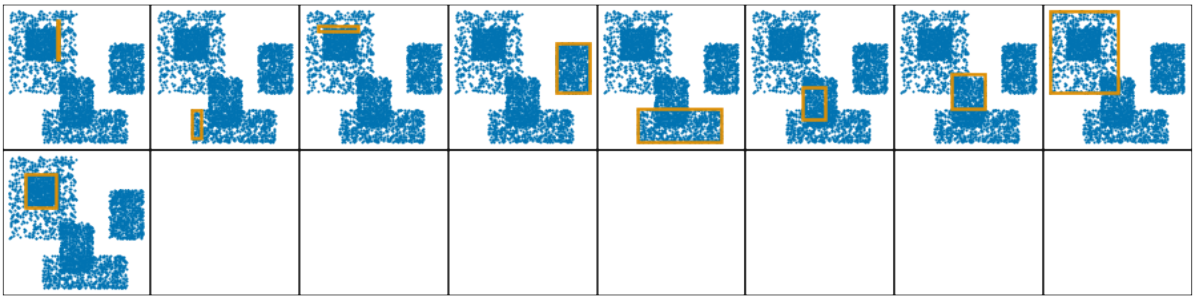
(c) REALKRIMP (sampling of size $0.5|G|$)

Figure 5.13: The results of pattern mining for "simple inclusion" dataset, support of the ground truth patterns is 200

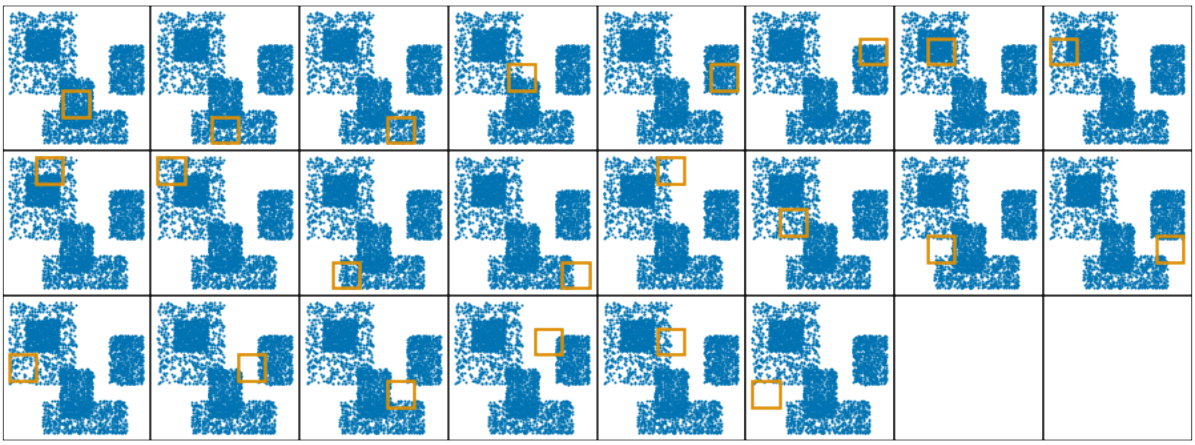
limitation of MINT. REALKRIMP return a simple description –the exact sparse pattern– as well as two noisy patterns.

Optimal number of intervals and neighbors. In MINT we use two parameters that can affect the results: the number of discretization intervals and the number of neighbors that are used to compute the candidates. For a dataset of $|G|$ objects and $|M|$ attributes, we consider the following numbers of dicretization intervals: 5, $0.5\sqrt{|G|}$, $\sqrt{|G|}$, $2\sqrt{|G|}$, and the following numbers of nearest neighbors: 5, $0.5\sqrt{|G|}$, $\sqrt{|G|}$, $2\sqrt{|G|}$, $0.5\sqrt{|G||M|}$, $\sqrt{|G||M|}$, $2\sqrt{|G||M|}$.

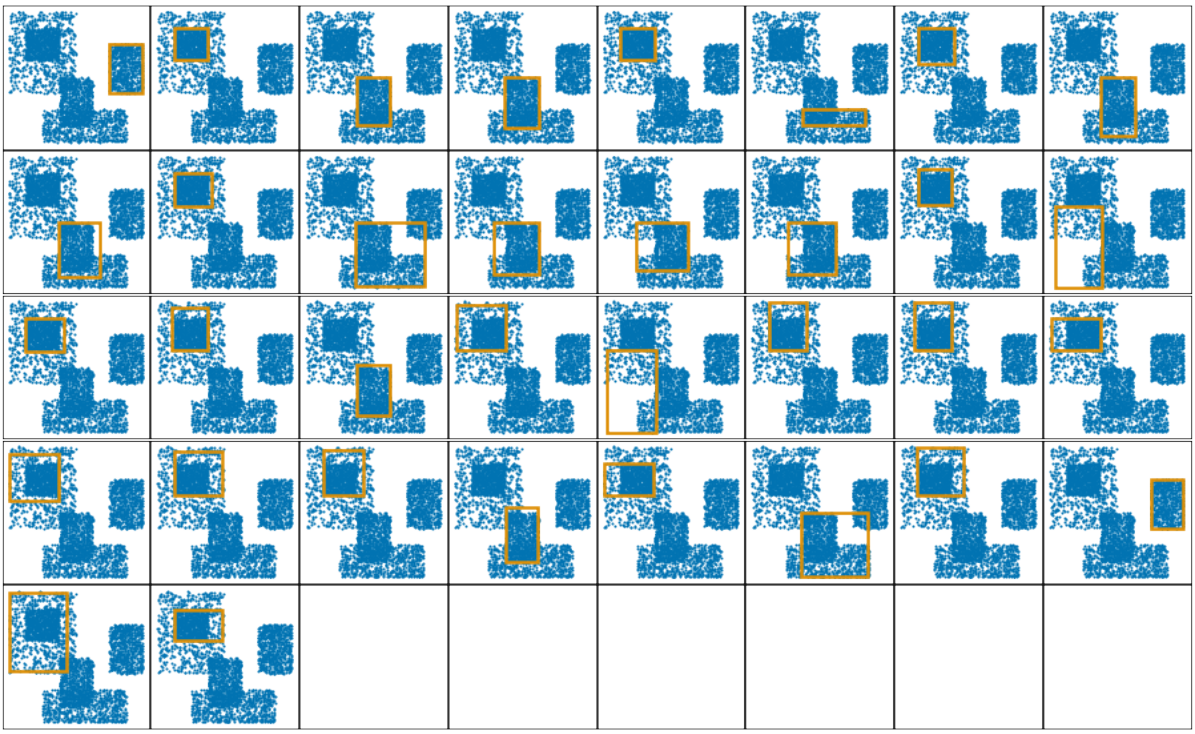
We evaluate the performance of MINT w.r.t. the aforementioned heuristics by the following characteristics: compression ratio, the number of patterns and the running time. The averages values of 9 real-world datasets ("iris", "waveform", "wine", "breast wisconsin", "spambase", "parkinsons", "haberman",



(a) MINT ($\sqrt{|G|}$)

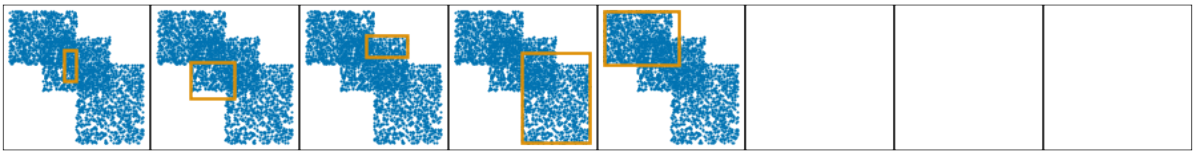


(b) SLIM (5)

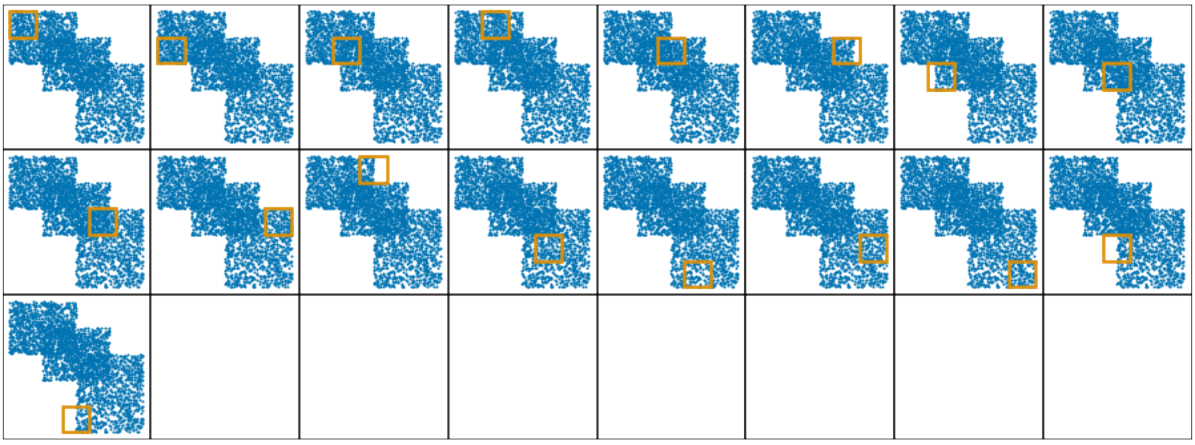


(c) REALKRIMP (sampling of size $0.5|G|$)

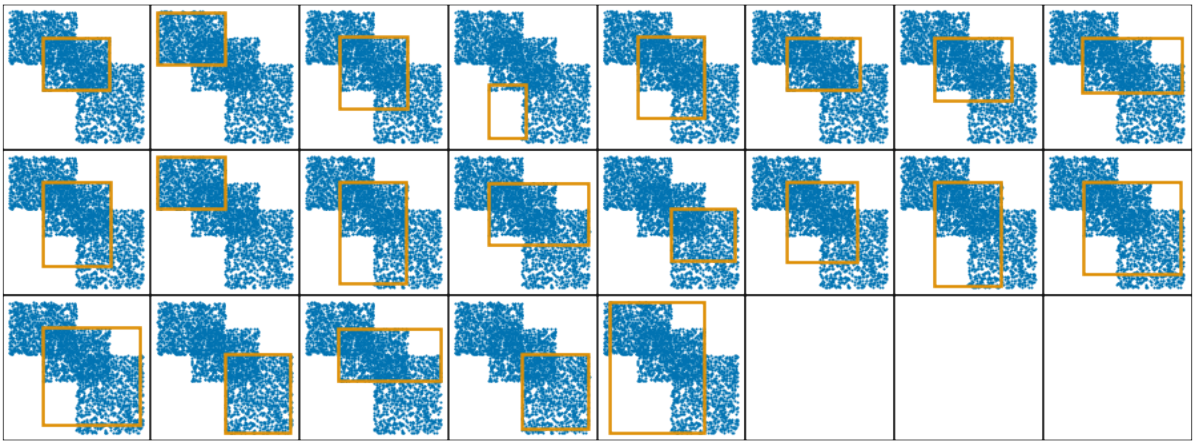
Figure 5.14: The results of pattern mining for "complex inclusion" dataset, support of the ground truth patterns is 700



(a) MINT ($\sqrt{|G|}$)



(b) SLIM (5)



(c) REALKRIMP (sampling of size $0.5|G|$)

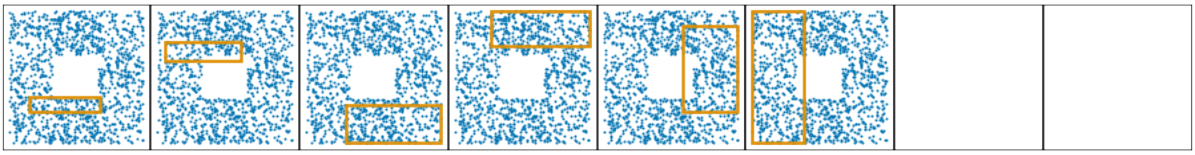
Figure 5.15: The results of pattern mining for "simple overlaps" dataset, support of the ground truth patterns is 1000

"ecoli", "statlog satimage") are given in Fig. 5.17. We also considered 6 types of synthetic datasets (see Fig. 5.7), where each pattern has support 100, 200, 500, 700, and 1000 (i.e., 30 synthetic datasets in total). The average values for the synthetic datasets are given in Fig. 5.18.

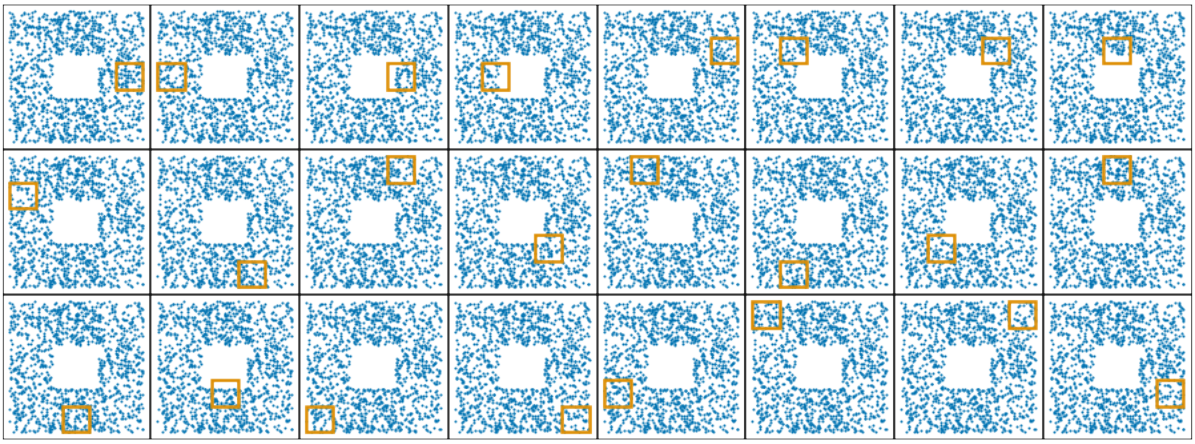
The results of experiments show that a very coarse discretization is less suitable for MINT since the compression ratio and the number of patterns is the largest (worst).

For fine discretized datasets the number of neighbors does not affect a lot the results. It appears that with a large number of discretized intervals, the results of MINT is less affected by the chosen heuristics. Taking into account the running time, the best (and the most "stable") results are achieved by splitting the attribute range into $\sqrt{|G|}$ intervals and by taking $\sqrt{|G|}$ nearest neighbors to compute pattern candidates.

We also study how useful is the pruning strategy for reducing the compression ratio and the number of patterns, and measure how much time it takes. The average compression gain, the number of removed



(a) MINT ($\sqrt{|G|}$)



(b) SLIM (5)

(c) REALKRIMP (sampling of size $0.5|G|$)

Figure 5.16: The results of pattern mining for "inverted" dataset, the number of data points is 200

patterns, and the time of pruning are given in Fig. 5.19 Fig. 5.20, for real-world and synthetic datasets, respectively. The results show that the pruning strategy is useful for mining hyper-rectangles in datasets with a coarse discretization. For fine discretized datasets, the compression ratio and the number of patterns are reduced only slightly.

MINT vs IPD. In the previous sections we discussed an MDL-based approach for data discretization called IPD. Like MINT, it relies on the MDL principle and deals with a fine discretization. Instead of a pattern set, IPD computes a discretization grid. However, these tasks look very similar, and it may appear both MINT and IPD perform quite the same task. Nevertheless, there is a principle conceptual difference between them. IPD is meant to compute a global discretization grid, i.e., to split the whole multi-dimensional space with a minimal number of global splittings of attribute ranges by preserving the interaction between attributes.

As it was already mentioned in Section 5.2.1, the methods for computing the global grid, in particular the MDL-based ones, often tend to build this grid with a small number of splittings. When the boundaries of patterns are not well-aligned, a small number of splitting does not allow to obtain good results: the description of patterns will be imprecise. We illustrate this problem using one of the synthetic datasets called "complex inclusion". The coarsest global discretization grid sufficient to describe precisely all patters is given in Fig. 5.21 (left). However, with the methods meant to compute "optimal global splitting" we hardly ever get a grid that allows us to reconstruct precisely all patterns. The IPD discretizer, for example, returns the grid where splittings are located in "approximately correct" positions: where several patterns are well aligned, IPD returns a correct splitting, while when several "correct" splittings are located next

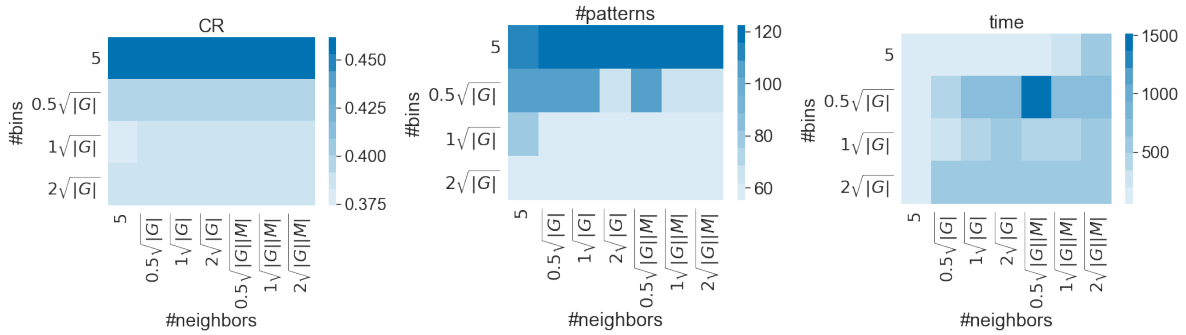


Figure 5.17: Average compression ratio, number of patterns, and running time of real-world dataset for different combinations of the parameters of MINT

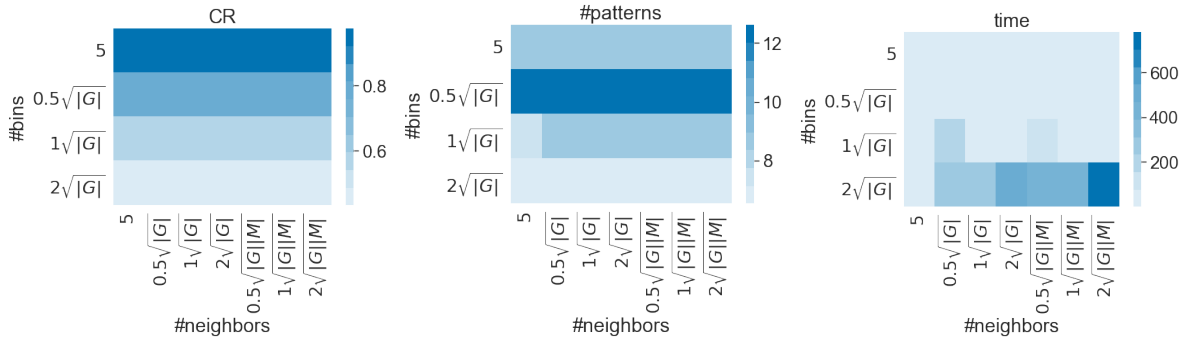


Figure 5.18: Average compression ratio, number of patterns, and running time of real-world (top) and synthetic (bottom) dataset for different combinations of the parameters of MINT

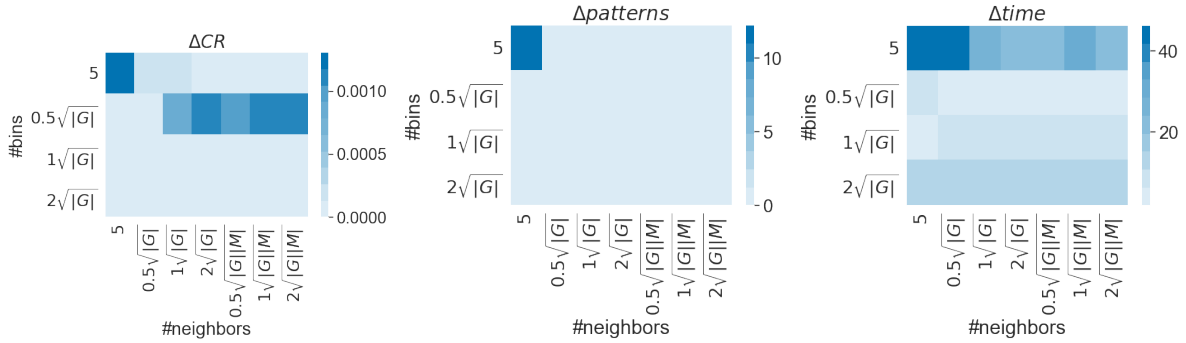


Figure 5.19: Average gains in compression ratio and in the number of patterns for real-world datasets achieved by applying MINT-PRUNING. The running time of MINT-PRUNING is reported on the right figure

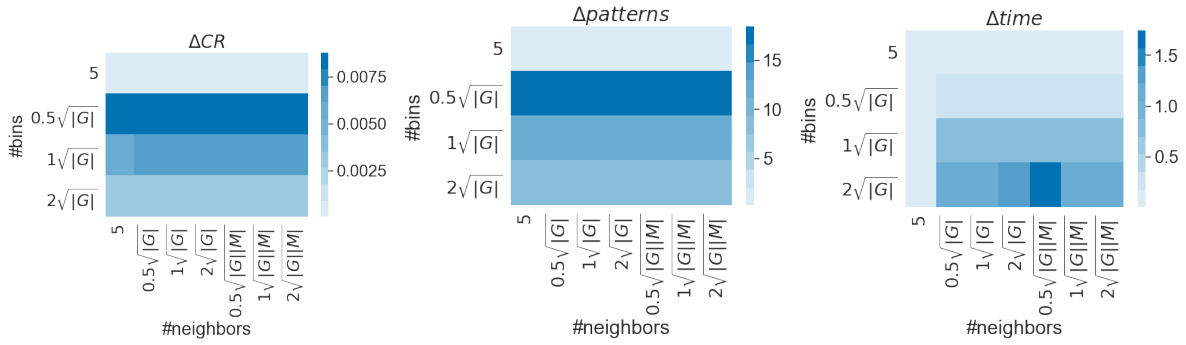


Figure 5.20: Average gains in compression ratio and in the number of patterns for real-world (top) and synthetic (bottom) datasets achieved by applying MINT-PRUNING. The running time of MINT-PRUNING is reported on the right figure

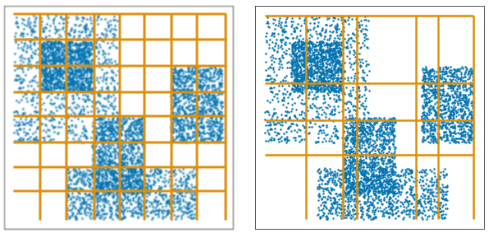


Figure 5.21: "Complex inclusion" datasets with a coarsest grid allowing for reconstruction all patterns precisely (left), IPD-based discretization (middle), examples of imprecise elementary hyper-rectangles obtained by IPD.

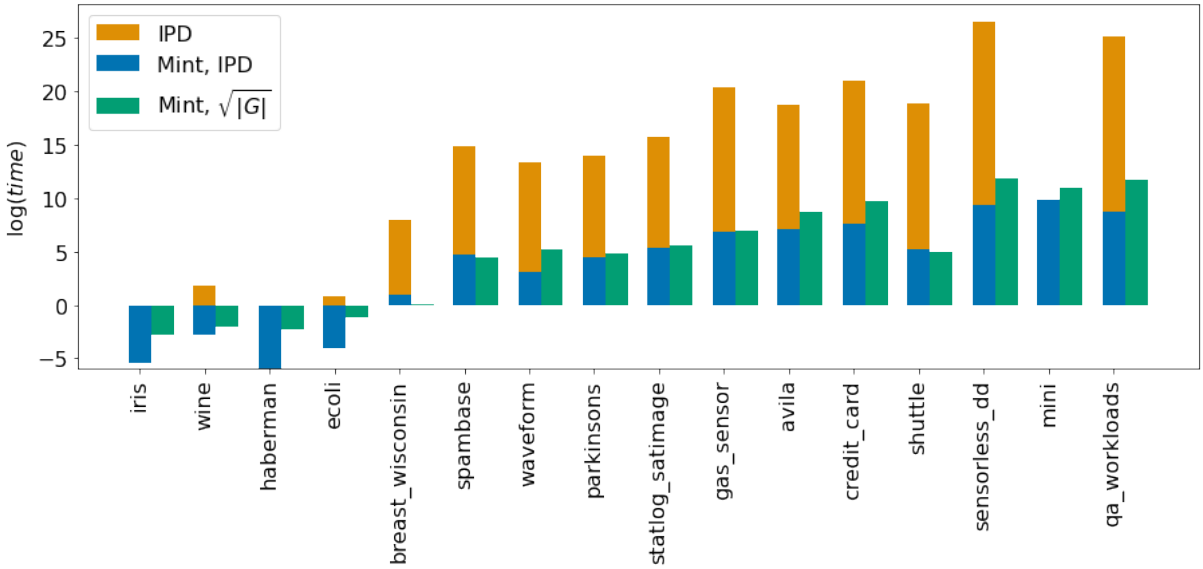


Figure 5.22: The execution time in seconds in logarithmic scale of Mint (applied to the dataset where each attribute is discretized into $\sqrt{|G|}$ bins) and IPD + Mint (IPD applied to the dataset where each attribute is discretized into $\sqrt{|G|}$ bins, while Mint is applied to the dataset discretized with IPD. The datasets are ordered by the number of objects they contain (in ascending order)

to each other, IPD often returns one that is located somewhere between the "correct" ones.

As a consequence, we either obtain elementary hyper-rectangles containing only a small fragment of pattern, or even a hyper-rectangles containing a mix of different ground truth patterns. We studied in detail the problem of imprecise pattern descriptions in the previous sections.

Apart from this issue, we also consider the running time of Mint and IPD. As was mentioned above, IPD returns only a global grid, thus to compute hyper-rectangles, a pattern mining method should be applied to merge further the elementary hyper-rectangles induced by the global grid. In particular, we may apply Mint for this purpose. However, Mint can be applied to the same dataset as IPD to mine directly hyper-rectangles.

In Fig. 5.22 we show the running time of Mint (computed on datasets where each attribute is discretized into $\sqrt{|G|}$ elementary intervals) and IPD (computed on datasets with the same fine discretization) with the consecutive application of Mint.

The results show that IPD works much slower than Mint, especially for large datasets. Moreover, taking into account the necessity to apply a pattern mining after, the running time of the pipeline "IPD + a pattern mining" is much larger than the application of Mint to fine discretized datasets. Thus, considering both the quality of hyper-rectangles and the running time allows us to conclude that Mint provides much better results than IPD in a shorter time.

5.6 Discussion and conclusion

Numerical pattern mining is a challenging task that is sometimes compared to clustering. However, even if the two tasks may appear similar, in numerical pattern mining the shapes of patterns are of first importance while in clustering the main concern is the similarity between objects and the resulting groups of objects. Accordingly, in this chapter we propose a formalization of numerical pattern set mining problem based on the MDL principle and we focus on the following characteristics: (i) interpretability of patterns, (ii) precise pattern descriptions, (iii) non-redundancy of pattern sets, and (iv) scalability. We study and materialize these characteristics, and we also propose a working implementation within a system called Mint.

By "interpretability" we mean not only the ability to explain why a particular pattern is selected but also the ease of analyzing a set of discovered numerical patterns for a human agent. With this regard, patterns of arbitrary shapes (e.g., [FvL20]) or even polygons (e.g., [BKRK17]) may be not an appropriate choice when considering multidimensional numerical data. That is why we decided to work with one of the most common shapes, namely "hyper-rectangles", which are currently used in numerical pattern mining and related tasks [WDKG14, KKN11b].

Another important requirement is that the boundaries of patterns should be "well-defined" and "quite precise". A common approach to numerical pattern mining consists in firstly a data binarization and secondly a reduction to itemset mining. Such an approach suffers from various drawbacks among which (i) the boundaries of patterns are not well-defined and this heavily affects the output, (ii) the scalability is not good because of the potentially exponential number of attributes due to scaling, (iii) the information loss related to the loss of the interval order within a range may be very important... In our experiments we compare the behavior of Mint with the MDL-based itemset set miner Slim (associated with a scaling of numerical data). The experiments demonstrate that Slim generally provides quite poor patterns. Actually, when the discretization is too fine, Slim is not able to merge patterns into large patterns, while when the discretization is too coarse the algorithm returns very imprecise boundaries. In addition, we also consider another MDL-based algorithm, namely RealKrimp, which is, to the best of our knowledge, the only MDL-based approach dealing with numerical pattern mining without any prior data transformation. However, one main drawback of RealKrimp is that it mines patterns individually, and then the resulting patterns are very redundant.

Furthermore, in the experiments, both RealKrimp and Slim show a poor scalability. Mint may also have a high running time for some large datasets, but still staying at a reasonable level.

Mint may appear to be similar to IPD –for "Interaction-Preserving Discretization"– but both systems perform different tasks. Mint could work in combination with IPD since the latter does not return exactly patterns but mainly MDL-selected boundaries. The elementary hyper-intervals induced from IPD results are only fragments of ground truth patterns. Then Mint could be applied to merge these elementary hyper-rectangles into larger hyper-rectangles.

Indeed, our experiments show that the data compressed by IPD can be even more compressed in applying Mint, i.e., the patterns as computed by IPD should still be completed for being comparable to those discovered by Mint. However, as the experiments show it, directly applying Mint to fine discretized data allows obtaining better results than applying IPD as a preprocessing step. This can be explained by the fact that IPD returns uniform or global boundaries, which are less precise than the boundaries specifically "tuned" by Mint for each pattern.

For summarizing, the Mint algorithm shows various very good capabilities w.r.t. its competitors, among which a good behavior on fine discretized datasets, good scalability, and it outputs a moderate number of non-redundant patterns with precise boundaries. However, there is still room for improving Mint, for example in avoiding redundant patterns and in the capability of mining sparse regions in the same manner as dense ones.

Future work may be followed in several directions. Here, Mint works with an encoding based on prequential plug-in codes. It could be interesting to reuse another encoding and to check how the performance of the system evolves, trying to measure what is the influence of the encoding choice. Moreover, we should consider more datasets and especially large and complex datasets, and try to measure the limit of the applicability of Mint, in turn improving the algorithm in the details. In general, more experiments should still be considered for improving the quality of Mint. Another interesting future

direction is to use MINT in conjunction with clustering algorithms. This could be a good way of associating descriptions or patterns with the classes of individuals that are discovered by a clustering process. In this way, a description in terms of attribute and ranges of values could be attached to the discovered clusters and complete the definition of the set of individuals which are covered. This could be reused in ontology engineering for example, and as well in numerous tasks where clustering is heavily used at the moment.

6

Conclusion and future work

Pattern mining is one of the seminal fields of data analysis. Since the first studies, the methods and approaches to pattern mining have been continuously improved: starting from the interestingness measures for single itemsets to the methods for mining pattern sets in complex data. The modern approaches are based on a solid theoretical foundation and advanced techniques for incorporation of background knowledge into the mining process.

Nowadays, there exists a large variety of methods for mining different patterns for very different data types. Despite the extensive work in this field, some important problems remain under-developed. In this study, we go back to the basics and revise the essential issues of pattern mining in the context of simple data types: binary and numerical tabular data.

Data topology. Data and pattern complexity. The first part of this study is focused on the theoretical aspects of pattern mining in binary data. We go beyond the notion of interestingness and study the intrinsic structure underlying the data. This structure does not rely on any subjective measures or hypotheses about pattern interestingness, but it utilises so-called equivalence classes of patterns in a given dataset. To define this structure, we used the framework of Formal Concept Analysis that provides strong mathematical tools for dealing with the equivalence classes. We introduced the *passkeys*, which are the simplest (minimum) elements in the equivalence classes.

Based on the passkeys, we introduced the *closure structure*, which is the theoretical concept allowing us to structure the equivalence classes into the levels w.r.t. the "itemset complexity". The size of a passkey defines the level of the closure structure to which the corresponding equivalence class belongs. We showed theoretical results about the ordinal nature of the closure structure, and we stated and proved complexity results about the computation of passkeys.

We also proposed the GDPM algorithm and its variations for computing the closure structure by levels with polynomial delay. The application of GDPM over a collection of datasets showed a rather efficient behavior of the algorithm. It is important to stress that knowing the level in the closure structure to which a closed itemset (equivalence class) belongs — it is obtained by GDPM — allows reducing drastically the complexity of the passkey enumeration problem.

The closure structure can be of benefit to practitioners. It allows for flexible representation of the "data topology" w.r.t. an interestingness measure and also allows for a concise representation of the "non-redundant" implications.

For synthesizing, the closure structure of a dataset brings the following benefits:

- The closure structure is determined by the closed itemsets and their equivalence classes. It is based on a partition related to the smallest keys in an equivalence class, called passkeys.
- The closure structure determines a "topology" of a dataset and is organized around a set of levels depending on the size of the passkeys. This provides a distribution of the itemsets in terms of their associated passkeys. Moreover, the number of levels, called the closure index, can be used to measure the potential size of the pattern space, i.e., given by the minimal Boolean lattice included in the pattern space.

- Given the closure structure and the closure levels, we can understand how itemsets are distributed by levels and know when to stop the search for interesting itemsets with a minimal loss of information. For example, we discovered that the first levels of the closure structure have better characteristics than the other levels. In particular, the passkeys of the first levels provide an almost complete covering of the whole dataset under study and allow to derive the implications with the smallest antecedents and the maximal consequents, which are considered as the best rules to be discovered in a dataset.

However, the proposed closure structure has some limitations and leaves much room for further research. We sketch some of these future investigations below.

- We anticipate that the developing formalism may be fruitful for learning data structure based on the neural networks [DHS19, MKC20], i.e., simply-derivable first levels of the closure structure contain the information that may improve the performance of the tasks on the prediction of some structural characteristics of the concept lattices.
- Considering complex datasets, one possible direction is to use the formalism of pattern structures, which is an extension of FCA allowing to directly work on complex data such as numbers, sequences, trees, and graphs [KKND11].
- Another important direction towards generalization of the closure structure is its adaptation for dealing with noisy data, in particular, not only with exact tiles (formal concepts) and implications but also with dense tiles (biclusters) and association rules.
- The closure structure is introduced and studied thanks to frequency which depends on the support of itemsets. Along with frequency, it could be very interesting to reuse other measures that would bring different and complementary views on the data, such as e.g., stability, lift, and MDL.
- The closure structure is intended in this thesis in terms of pattern mining, and more precisely in terms of itemset mining. The question of the generalization can be raised, for example how the closure structure could be reused in terms of investigations related to pattern mining, such as subgroup discovery, anytime approaches in pattern mining, and also, as already mentioned above, MDL-based pattern mining?
- Many questions remain when we consider the practical point of view. Among these questions, how to define a possibly anytime strategy for pattern mining terms based on the closure structure, knowing in particular that the first levels are the most interesting. Moreover, this strategy could also guide the discovery of the most interesting implications and association rules, and take into account the relations with functional dependencies.
- We also emphasize the "brittleness" of the closure structure and the passkeys. The latter means that as new data become available, the passkeys of a given closed itemset can be very different from those that were computed on a fragment of the dataset. This question is closely related to the "stability" of formal concepts [Kuz90] and Δ-measure [BEJ$^+$13]. The relation between them remains an important question to study. We anticipate that taking into account the neighborhood of a formal concept (closed itemset) in the concept lattice, including their passkeys, may shed the light on the stability of the passkeys.
- The last direction we mention here is the improvement of the algorithms for the passkey enumeration. In particular, we deem that exploiting the enumeration strategies used in Talky-G or DefMe may improve the performance of GDPM.

Itemset mining. Another direction of this study was devoted to itemset and association rule mining, where the notion of interestingness is fundamental. However, this notion is subjective, and there exists a large variety of methods and approaches to define interestingness, and the results obtained by these methods may be very different. The lack of ground truth complicates the process of learning from data. That is why the background knowledge or even intuition of experts often play a crucial role in practice.

A lot of modern approaches are based on the MDL principle, which is often referred to as *learning by compression*. According to the MDL principle, the patterns are considered to be interesting if they compress well. Thus, in the MDL-based approaches, the problem of mining interesting patterns is transformed into searching for some regularities in data allowing for good compression.

However, apart from evaluating interestingness (by *interestingness measures*), we take into account *quality measures* that formalize very intuitive expectation from the results, i.e., non-redundancy, good descriptiveness, compactness, etc.

In this study, we addressed some important issues that follow directly from the aforementioned specificity of itemset and association rule mining:

- pattern explosion problem;
- a possible discrepancy between the subjective notion of interestingness and common intuition about the quality of the pattern sets.

Pattern explosion is a common problem for pattern mining approaches. We proposed an approach for reducing the itemset search space relying on a variation of the independence model and assuming that a pattern is interesting if it appears more often than it was expected. We emphasize that this approach may replace frequency-based enumeration algorithms, however, if there exists an algorithm that is tailored for the specific interestingness measure, it is preferable to use it.

Regarding the second issue — compliance of the interestingness measure with the quality measures — we revise the problem of the lack of ground truth in pattern mining. Based on the assumption that the best techniques from supervised learning with the available ground truth, which work generally well, may work well even when no ground truth is available.

We propose the KeepItSimple algorithm that exploits some ideas that are widely used for constructing the decision tree ensembles: biclosed itemsets (an analog of shallow trees and stumps), multi-model description with the projections (an analog of boosted trees), and modification of the description length introduced in [VVLS11] (for the conformity of the interestingness measure with the quality measure and to make the itemset estimates less dependent on the heuristics). Our experiments shows that KeepItSimple improves the results of Krimp w.r.t. very intuitive quality measures.

The results of the experiments suggest that exploiting the state-of-the-art techniques from supervised learning may reduce the gap between "quality" and "interestingness" of pattern sets. The latter demonstrates great potential for developing methods involving background knowledge or the best practices from the other fields to obtain better results of pattern mining. Development of the approaches not only for itemsets but also for other pattern and data types would be a very natural extension of this work.

Numerical pattern mining. The last important direction of this study is numerical pattern mining. Numerical pattern mining is a challenging task that is sometimes compared to clustering. However, even if these two tasks may appear similar, in numerical pattern mining the shapes of patterns are of first importance while in clustering the main concern is the similarity between objects and the resulting groups of objects.

Although numeric data is a common data type, numerical pattern mining remains under-developed. The most important related problem is data preprocessing. For numerical data one usually performs two preprocessing steps: discretization and/or binarization. In this study we address the problem of prepossessing as well and study how such factors as (i) imprecise boundaries caused by a globally "optimal" grid, (ii) the information loss caused by binarization, and (iii) attribute explosion may hamper the pattern mining process.

Accordingly, we proposed a formalization of numerical pattern set mining problem based on the MDL principle and we focus on the following characteristics: (i) interpretability of patterns, (ii) precise pattern descriptions, (iii) non-redundancy of pattern sets, and (iv) scalability.

By "interpretability" here we mean not only the ability to explain why a particular pattern is selected but also the ease of analyzing a set of discovered numerical patterns for a human agent. With this regard, patterns of arbitrary shapes (e.g., [FvL20] for binary data) or even polygons (e.g., [BKRK17]) may be not an appropriate choice when considering multidimensional numerical data. That is why we decided to work with one of the most common shapes, namely "hyper-rectangles", which are currently used in numerical pattern mining and related tasks [WDKG14, KKN11b].

We propose an efficient MDL-based approach, called MINT, that computes a small set of non-redundant and good-quality hyper-rectangles.

In our experiments we compare the behavior of MINT with the MDL-based itemset set miner SLIM (associated with a scaling of numerical data). The experiments demonstrate that SLIM generally provides quite poor patterns. Actually, when the discretization is too fine, SLIM is not able to merge patterns into large patterns, while when the discretization is too coarse the algorithm returns very imprecise boundaries. In addition, we also consider another MDL-based algorithm, namely REALKRIMP, which is, to the best of our knowledge, the only MDL-based approach dealing with numerical pattern mining without any prior data transformation. However, one main drawback of REALKRIMP is that it mines patterns individually, and then the resulting patterns are very redundant.

Furthermore, in the experiments, both REALKRIMP and SLIM show a poor scalability. MINT may also have a high running time for some large datasets, but still staying at a reasonable level.

MINT may appear to be similar to IPD — for "Interaction-Preserving Discretization" — but both systems perform different tasks. MINT could work in collaboration with IPD since the latter does not return exactly patterns but mainly MDL-selected boundaries. The elementary hyper-intervals induced from IPD results are only fragments of ground truth patterns. Then MINT could be applied to merge these elementary hyper-rectangles into larger hyper-rectangles.

Indeed, our experiments show that the data compressed by IPD can be even more compressed in applying MINT, i.e., the patterns as computed by IPD should still be completed for being comparable to those discovered by MINT. However, as the experiments show it, directly applying MINT to fine discretized data allows obtaining better results than applying IPD as a preprocessing step. This can be explained by the fact that IPD returns uniform or global boundaries, which are less precise than the boundaries specifically "tuned" by MINT for each pattern.

For summarizing, the MINT algorithm shows various very good capabilities w.r.t. its competitors, among which a good behavior on fine discretized datasets, good scalability, and it outputs a moderate number of non-redundant patterns with precise boundaries. However, there is still room for improving MINT, for example in avoiding redundant patterns and in the capability of mining sparse regions in the same manner as dense ones.

Future work may be followed in several directions.

- Here, MINT works with an encoding based on prequential plug-in codes. It could be interesting to reuse another encoding and to check how the performance of the system evolve, trying to measure what is the influence of the encoding choice.

- Moreover, we should consider more datasets and especially large and complex datasets, and try to measure the limit of the applicability of MINT, in turn improving the algorithm in the details. In general, more experiments should still be considered for improving the quality of MINT.

- MINT discretizes data under the hood. The experiments showed that fine discretization is the most suitable for MINT, however, the question on formal evaluation of the best discretization parameters w.r.t. a particular dataset remains open. Another possible direction consists in improving heuristics of MINT or developing the techniques for elimination of unimportant attributes from a pattern description (as it is done in REALKRIMP).

- Another interesting future direction is to use MINT in conjunction with clustering algorithms. This could be a good way of associating descriptions or patterns with the classes of individuals that are discovered by a clustering process. In this way, a description in terms of attribute and ranges of values could be attached to the discovered clusters and complete the definition of the set of individuals which are covered. This could be reused in ontology engineering for example, and as well in numerous tasks where clustering is heavily used at the moment.

List of publications

Conferences

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. Gradual Discovery with Closure Structure of a Concept Lattice. In *Proceedings of the 15th International Conference on Concept Lattices and Their Applications, CLA 2020, Tallinn, Estonia, June 29-July 1, 2020*, pages 145–157.

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. On Coupling FCA and MDL in Pattern Mining. In *Proceedings of the 15th International Conference on Formal Concept Analysis, ICFCA 2019, Frankfurt, Germany, June 25-28, 2019*, pages 332–340.

- Tatiana Makhalova, Martin Trnecka. A Study of Boolean Matrix Factorization Under Supervised Settings. In *Proceedings of the 15th International Conference on Formal Concept Analysis, ICFCA 2019, Frankfurt, Germany, June 25-28, 2019*, pages 341–348.

- Tatiana Makhalova, Dmitry Ilvovsky, Boris Galitsky. Information Retrieval Chatbots Based on Conceptual Models. In *Proceedings of the 24th International Conference on Conceptual Structures, ICCS 2019, Marburg, Germany, July 1–4, 2019*, pages 230–238.

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. Numerical Pattern Mining Through Compression. In *Proceedings of the Data Compresion Conference, DCC 2019, Snowbird, Utah, USA, March 26-29, 2019*, pages 112–121.

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. A First Study on What MDL Can Do for FCA. In *Proceedings of the 14th International Conference on Concept Lattices and Their Applications, CLA 2018, Olomouc, Czech Republic, June 12-14, 2018*, pages 25–36.

- Tatiana Makhalova, Sergei O. Kuznetsov. On overfitting of classifiers making a lattice. In *Proceedings of the 14th International Conference on Formal Concept Analysis, ICFCA 2017, Rennes, France, June 13-16, 2017*, pages 184–197.

- Tatiana Makhalova, Sergei O. Kuznetsov. On Stability of Triadic Concepts. In *Proceedings of the 13th International Conference on Concept Lattices and Their Applications, CLA 2016, Moscow, Russia, July 18-22, 2016*, pages 245–253.

- Tatiana Makhalova, Sergei O. Kuznetsov. Concept Interestingness Measures: a Comparative Study. In *Proceedings of the 12th International Conference on Concept Lattices and Their Applications, CLA 2015, Clermont-Ferrand, France, October 13–16, 2015*, pages 59–72.

Workshops

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. Adaptive Multi-model Approaches to Pattern Set Mining. In *Proceedings of the 9th European Starting AI Researchers' Symposium 2020 co-located with 24th European Conference on Artificial Intelligence (ECAI 2020), Santiago Compostela, Spain, August, 2020*, 8 pages.

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. Closure Structure: a Deeper Insight. in *Proceedings of the 8th International Workshop "What can FCA do for Artificial Intelligence?" (FCA4AI 2020) co-located with 24th European Conference on Artificial Intelligence (ECAI 2020), Santiago de Compostela, Spain, August 29, 2020*, pages 45–56.
- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. MDL for FCA: is there a place for background knowledge? In *Proceedings of the 6th International Workshop "What can FCA do for Artificial Intelligence?" co-located with International Joint Conference on Artificial Intelligence and European Conference on Artificial Intelligence, FCA4AI@IJCAI 2018, Stockholm, Sweden, July 13, 2018*, pages 45–56.
- Tatiana Makhalova, Lhouari Nourine. An Incremental Algorithm for Computing n-concepts. In *Proceedings of International Workshop on Formal Concept Analysis for Knowledge Discovery, FCA4KD 2017, Moscow, Russia, 1 June, 2017*, pages 43–52.
- Tatiana Makhalova, Dmitry Ilvovsky, Boris Galitsky. Pattern Structures for News Clustering. In *Proceedings of the 4th International Workshop "What can FCA do for Artificial Intelligence?" co-located with the International Joint Conference on Artificial Intelligence, FCA4AI@IJCAI 2015, Buenos Aires, Argentina, July 25, 2015*, pages 35–42.
- Tatiana Makhalova, Dmitry Ilvovsky, Boris Galitsky. News clustering approach based on discourse text structure. In *Proceedings of the First Workshop on Computing News Storylines, CNS@ACL-IJCNLP 2015, Beijing, China, July 26-31*, pages 16–20.

Journals

- Tatiana Makhalova, Martin Trnecka. From-below boolean matrix factorization algorithm based on MDL. *Advances in Data Analysis and Classification*, 1–20, 2020.
- Tatiana Makhalova, Sergei O. Kuznetsov. On interestingness measures of formal concepts. *Information Sciences*, 442: 202–219, 2018.

Preprints

- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli, A. Buzmakov. Discovery data topology with the closure structure. Theoretical and practical aspects, arXiv: 2010.02628, 2020.
- Tatiana Makhalova, Sergei O. Kuznetsov, Amedeo Napoli. Mint: MDL-based approach for Mining INTeresting Numerical Pattern Sets, arXiv: 2011.14843, 2020.

Detailed summary in French

Dans cette thèse, nous étudions différents aspects de l'exploration ou fouille de motifs dans des jeux de données tabulaires (binaires et numériques). L'objectif de l'exploration de motifs est de découvrir un petit ensemble de motifs non redondants qui peuvent recouvrir presque entièrement un ensemble de données. Ces motifs peuvent être interprétés comme des unités de connaissances significatives et utiles. Nous nous concentrons sur certaines questions clés telles que :

- la définition formelle de l'intérêt des motifs,
- la minimisation de l'explosion combinatoire des motifs,
- la définition de mesures permettant d'évaluer les performances des méthodes d'exploration de motifs, et
- le rapprochement entre l'intérêt et la qualité des ensembles de motifs.

De plus, nous allons au-delà des investigations classiques sur l'exploration de motifs en faisant émerger et en étudiant la structure intrinsèque sous-jacente à un jeu de données tabulaire. Les principales contributions de ce travail de recherche sont à la fois théoriques, conceptuelles et pratiques.

L'exploration de modèles est l'un des domaines fondateurs de l'analyse de données. Depuis les premières études, les méthodes et les approches de l'exploration de motifs ont été continuellement améliorées : des mesures d'intérêt applicable à chaque motif aux méthodes d'exploration d'ensembles de motifs.

Les approches modernes sont basées sur un fondement théorique solid et des techniques avancées pour l'incorporation de connaissances de base dans le processus d'extraction.

De nos jours, il existe une grande variété de méthodes permettant d'extraire différents modèles pour des types de données très différents. Malgré le travail considérable dans ce domaine, certains problèmes importants restent cependant sous-développés. Dans cette étude, nous revenons aux fondamentaux et révisons les enjeux essentiels de l'exploration de motifs pour les données simples : les données tabulaires binaires et numériques.

Chapitre 2. Notions de base

Dans ce chapitre, nous décrivons les concepts de base utilisés tout au long de la thèse. Tout d'abord, dans la section 2.1, nous introduisons les notions fondamentales de la fouille de motifs dans les données binaires dans le cadre de l'analyse formelle de concept (FCA) [GW99].

Ce cadre fournit un formalisme très générique pour traiter les itemsets, les tuiles (tiles) et d'autres types de motifs dans les données binaires.

De plus, il permet de structurer l'espace de recherche de motifs en classes de "motifs équivalents".

La force du formalisme de la FCA est qu'il fournit non seulement des outils uniformes pour traiter un large spectre de motifs dans des données binaires, mais qu'il peut également être facilement généralisé à des données plus complexes, en particulier à des données numériques.

Dans la section 2.2, nous présentons des structures de motifs d'intervalles, qui sont une généralisation de la FCA au cas des données numériques. Dans le chapitre 5, nous proposons une approche pour extraire un ensemble de concepts de motifs d'intervalles.

Après avoir présenté les modèles binaires et numériques que nous utilisons dans cette étude, nous introduisons dans la section 2.3 les bases du principe de longueur minimale de description (MDL), que nous utilisons pour extraire des modèles binaires et numériques.

Enfin, dans la section 2.4, nous donnons un aperçu rapide des approches existantes pour évaluer les résultats de l'exploration de motifs. Nous n'abordons dans cette section que des notions très générales, certaines mesures de qualité spécifiques sont introduites dans les sections où nous les utilisons.

Chapitre 3. Structure de niveaux de fermetures

La première partie de cette étude est centrée sur les aspects théoriques de l'exploration de motifs dans les données binaires. Nous dépassons la notion d'intérêt et étudions la structure intrinsèque sous-jacente aux données. Cette structure ne repose sur aucune mesure ou hypothèse subjective concernant l'intérêt des modèles, mais elle utilise des classes d'équivalence de modèles dans un ensemble de données. Pour définir cette structure, nous avons utilisé le cadre de l'analyse de concept formelle qui fournit des outils mathématiques solides pour traiter les classes d'équivalence. Nous avons introduit les passkeys, qui sont les éléments les plus simples (minimaux) des classes d'équivalence.

Sur la base des générateurs minimaux (clés), nous avons introduit la structure de niveaux de fermetures, le concept théorique nous permettant de structurer les classes d'équivalence dans les niveaux par rapport à la "complexité d'itemset". La taille d'un passkey définit le niveau de la structure de niveaux de fermetures auquel appartient la classe d'équivalence correspondante.

Nous avons montré des résultats théoriques sur la nature ordinale de la structure de niveaux de fermetures, et nous avons démontré des résultats de complexité sur le calcul des générateurs minimaux.

Nous avons également proposé l'algorithme GDPM et ses variants pour calculer la structure de niveaux de fermetures par niveaux avec un retard polynomial. L'application du GDPM sur une collection d'ensembles de données a montré un comportement plutôt efficace de l'algorithme. Il est important de souligner que connaître le niveau dans la structure de niveaux de fermetures auquel appartient un ensemble d'éléments fermé (classe d'équivalence) — il est obtenu par GDPM — permet de réduire drastiquement la complexité du problème d'énumération de générateurs minimaux.

La structure de niveaux de fermetures présente des avantages en pratique. Elle permet une représentation flexible de la "topologie des données" par rapport à une mesure d'intérêt et permet également une représentation concise des implications "non redondantes".

Pour résumer, la structure de niveaux de fermetures d'un jeu de données apporte les avantages suivants :

- La structure de niveaux de fermetures est déterminée par les itemsets fermés et leurs classes d'équivalence. Elle est basée sur une partition liée aux plus petits générateurs minimaux d'une classe d'équivalence, appelés passkeys.

- La structure de niveaux de fermetures détermine une "topologie" d'un ensemble de données et s'organise autour d'un ensemble de niveaux en fonction de la taille des passkeys. Cela fournit une distribution des ensembles d'éléments en termes de leurs passkeys associés. De plus, le nombre de niveaux, appelé indice de fermeture, peut être utilisé pour mesurer la taille potentielle de l'espace des motifs, c'est-à-dire donnée par le réseau booléen minimal inclus dans l'espace des motifs.

- La structure de niveaux de fermetures permet de comprendre comment les ensembles d'éléments sont distribués/répartis par niveaux et ainsi de savoir quand arrêter la recherche d'ensembles d'éléments intéressants en garantissant une perte d'information minimal. Par exemple, nous avons découvert que les premiers niveaux de la structure de niveaux de fermetures ont de meilleures propriétés que les autres niveaux. En particulier, les passkeys des premiers niveaux fournissent une couverture presque complète de l'ensemble de données et permettent d'en déduire les implications avec les plus petits antécédents et les conséquences maximales. Ces implications sont considérées comme les meilleures règles à découvrir dans un ensemble de données.

Cependant, la structure de niveaux de fermetures proposée présente certaines limites et et laisse de la place à de nombreuses recherches ultérieures. Nous en esquissons quelques-unes ci-dessous.

- Nous pensons que le formalisme en développement pourrait être profitable pour l'apprentissage de la structure de données basé sur les réseaux de neurones [DHS19, MKC20]. En effet, les premiers niveaux simplement dérivables de la structure de niveaux de fermetures contiennent les informations permettant d'améliorer la performance de la prédiction de certaines caractéristiques structurelles des treillis de concept.

- Considérant des jeux de données complexes, une direction possible est d'utiliser le formalisme des structures de patrons, une extension de FCA permettant de travailler directement sur des données complexes telles que des nombres, des séquences, des arbres et des graphes [KKND11].

- Une autre direction importante vers la généralisation de la structure de niveaux de fermetures est son adaptation pour traiter des données bruitées, en particulier avec des tuiles exactes (concepts formels) et des implications mais aussi avec des tuiles denses (biclusters) et des règles d'association.

- La structure de niveaux de fermetures est introduite et étudiée grâce à la fréquence, elle dépend du support des itemsets. Parallèlement à la fréquence, il pourrait être très intéressant de réutiliser d'autres mesures qui apporteraient des vues différentes et complémentaires sur les données, telles que stabilité, lift et MDL [Grü07].

- La structure de niveaux de fermetures est conçue dans cette thèse en termes d'extraction de motifs, et plus précisément en termes d'extraction d'itemsets. La question de la généralisation peut être soulevée, par exemple comment la structure de niveaux de fermetures pourrait être réutilisée dans des recherches liées à l'exploration de motifs, telles que la découverte de sous-groupes (subgroup discovery), les approches anytime dans l'exploration de motifs, et aussi, comme déjà mentionné ci-dessus, les modèles basés sur MDL?

- De nombreuses questions subsistent lorsque l'on considère le point de vue pratique. Parmi ces questions, comment définir une stratégie éventuellement anytime pour les termes d'exploration de motifs basée sur la structure de niveaux de fermetures, sachant notamment que les premiers niveaux sont les plus intéressants. De plus, cette stratégie pourrait également guider la découverte des implications et des règles d'association les plus intéressantes, et prendre en compte des relations ayant des dépendances fonctionnelles.

- Nous soulignons également la "fragilité" de la structure de niveaux de fermetures et des passkeys. Ce dernier signifie qu'à mesure que de nouvelles données deviennent disponibles, les passkeys d'un ensemble d'itemsets fermé donné peuvent être très différents de ceux qui ont été calculés sur un fragment de l'ensemble de données. Cette question est étroitement liée à la "stabilité" des concepts formels [Kuz90] et Δ-measure [BEJ+13]. La relation entre eux reste une question importante à étudier. Nous supposons que la prise en compte du voisinage d'un concept formel (ensemble d'itemsets fermés) dans le treillis de concepts peut faire la lumière sur la stabilité des passkeys.

- La dernière direction que nous mentionnons ici est l'amélioration des algorithmes pour l'énumération des passkeys. En particulier, nous estimons que l'exploitation des stratégies d'énumération utilisées dans Talky-G [SVN+14] ou DefMe [SR14] peut améliorer les performances du GDPM.

Chapitre 4. Exploration de motifs dans les données binaires

Une autre direction de cette étude a été consacrée à la fouille d'itemsets et de règles d'association, pour lesquelles la notion d'intérêt est fondamentale. Cependant, cette notion d'intérêt est subjective, et il existe une grande variété de méthodes et d'approches pour la définir, et les résultats obtenus par ces méthodes peuvent être très différents. Le manque de vérité terrain complique le processus d'apprentissage

à partir des données. C'est pourquoi les connaissances de base ou même l'intuition des experts jouent souvent un rôle crucial dans la pratique.

Beaucoup d'approches modernes sont basées sur le principe MDL [Grü07] : elles sont souvent appelées apprentissage par compression. Selon le principe MDL, les motifs sont considérés comme intéressants s'ils se compressent bien. Ainsi, dans les approches basées sur MDL, le problème de l'exploration de motifs intéressants se transforme en recherche de certaines régularités dans les données permettant une bonne compression.

Cependant, outre l'évaluation de l'intérêt (par des mesures d'intérêt), nous prenons en compte des mesures de qualité qui formalisent une attente très intuitive des résultats, comme par exemple la non-redondance, la bonne descriptivité, la compacité, etc.

Dans cette étude, nous avons abordé des questions importantes qui découlent directement de la spécificité susmentionnée de l'exploration d'ensembles d'éléments et de règles d'association :

- le problème de l'explosion des motifs,
- une divergence possible entre la notion subjective d'intérêt et l'intuition commune sur la qualité des ensembles de motifs.

L'explosion de motifs est un problème courant pour les approches d'exploration de motifs. Nous avons proposé une approche pour réduire l'espace de recherche des itemsets en s'appuyant sur une variation du modèle d'indépendance et en supposant qu'un motif est intéressant s'il apparaît plus souvent que prévu. Nous soulignons que cette approche peut remplacer les algorithmes de dénombrement basés sur la fréquence, cependant, s'il existe un algorithme adapté à la mesure d'intérêt spécifique, il est préférable de l'utiliser.

Concernant le deuxième problème nous revisitons le problème du manque de vérité terrain dans l'exploration de motifs. Nous nous basons sur l'hypothèse que les meilleures techniques d'apprentissage supervisé avec la vérité terrain disponible, qui fonctionnent généralement bien, peuvent bien fonctionner même lorsqu'aucune vérité terrain n'est disponible.

Nous proposons l'algorithme KeepItSimple qui exploite quelques idées largement utilisées pour construire les ensembles d'arbres de décision : itemsets biclos (un analogue d'arbres peu profonds), description multi-modèle avec les projections (un analogue d'arbres boostés), et modification de la longueur de description introduite dans [VVLS11] (pour la conformité de la mesure d'intérêt avec la mesure de qualité et pour rendre les estimations d'itemsets moins dépendantes de l'heuristique). Nos expériences montrent que KeepItSimple améliore les résultats de Krimp [VVLS11] par rapport à des mesures de qualité très intuitives.

Les résultats des expériences suggèrent que l'exploitation des techniques de pointe de l'apprentissage supervisé peut réduire l'écart entre la "qualité" et "l'intérêt" des ensembles de modèles. Cela démontre un grand potentiel pour développer des méthodes impliquant des connaissances de base ou les meilleures pratiques de l'apprentissage supervisé pour obtenir de meilleurs résultats de fouille de motifs. Le développement des approches pour les itemsets mais aussi pour d'autres types de patrons et de données serait une extension très naturelle de ce travail.

Chapitre 5. Exploration de motifs dans les données numériques

La dernière direction importante de cette étude est l'exploration de motifs numériques. L'exploration de motifs numériques est une tâche difficile qui est parfois comparée au clustering. Cependant, même si ces deux tâches peuvent sembler similaires, les formes des motifs sont de première importance dans l'exploration de motifs numériques, tandis que dans le clustering, la principale préoccupation est la similitude entre les objets et les groupes d'objets résultants.

Bien que les données numériques soient un type de données courant, l'exploration de motifs numériques reste sous-développée. Le problème connexe le plus important est le prétraitement des données. Pour les données numériques, on effectue généralement une ou deux étapes de prétraitement : la discrétisation et/ou la binarisation. Dans cette étude, nous abordons également le problème de la prétraitement et étudions comment des facteurs tels que

- des limites imprécises causées par une grille globalement "optimale",

– la perte d'informations causée par la binarisation, et

– l'explosion d'attributs qui peut entraver le processus d'extraction de motifs.

Par conséquent, nous avons proposé une formalisation du problème d'exploration de motifs numériques basée sur le principe MDL et nous nous concentrons sur les caractéristiques suivantes :

– l'interprétabilité des motifs,

– la description précises des motifs,

– non-redondance des motifs, et

– la scalabilite.

Par "interprétabilité", nous entendons ici non seulement la capacité d'expliquer pourquoi un motif particulier est sélectionné, mais aussi la facilité d'analyser par un humain un ensemble de motifs numériques découverts. À cet égard, des motifs de formes arbitraires (par exemple, [FvL20] pour les données binaires) ou même des polygones (par exemple, [BKRK17]) peuvent être un choix inapproprié lors de l'examen de données numériques multidimensionnelles. C'est pourquoi nous avons décidé de travailler avec l'une des formes les plus courantes, à savoir les "hyper-rectangles", qui sont actuellement utilisés dans l'exploration de motifs numériques et dans leurs tâches connexes [WDKG14, KKN11b].

Nous proposons une approche efficace basée sur MDL, appelée Mint, qui calcule un petit ensemble d'hyper-rectangles non redondants et de bonne qualité.

Dans nos expériences, nous comparons le comportement de Mint avec le pattern set miner basé sur MDL Slim (associé à une mise à l'échelle des données numériques).

Les expériences montrent que Slim fournit généralement des motifs assez médiocres. En effet, lorsque la discrétisation est trop fine, Slim n'est pas capable de fusionner des motifs en plus grands motifs, tandis que lorsque la discrétisation est trop grossière l'algorithme renvoie des frontières très imprécises. En outre, nous considérons également un autre algorithme basé sur MDL, à savoir RealKrimp [WDKG14], qui est, à notre connaissance, la seule approche basée sur MDL traitant de l'exploration de motifs numériques sans aucune transformation de données préalable. Cependant, un inconvénient principal de RealKrimp est qu'il extrait les motifs individuellement, et les motifs résultants sont alors très redondants.

De plus, dans les expériences, RealKrimp et Slim [SV12] montrent une faible scalabilité. Mint peut également avoir un temps d'exécution élevé pour certains grands ensembles de données, tout en restant à un niveau raisonnable.

Mint peut sembler similaire à IPD [NMVB14] — pour "Interaction-Preserving Discretization" — mais les deux systèmes effectuent des tâches différentes. Mint pourrait fonctionner en collaboration avec IPD puisque ce dernier ne renvoie pas exactement des motifs mais principalement des limites sélectionnées par MDL. Les hyper-intervalles élémentaires induits à partir des résultats de IPD ne sont que des fragments de motifs de vérité terrain. De ce fait, Mint pourrait être appliqué pour fusionner ces hyper-rectangles élémentaires en hyper-rectangles plus grands.

En effet, nos expériences montrent que les données compressées par IPD peuvent être encore plus compressées en appliquant Mint, c'est-à-dire que les motifs tels que calculés par IPD doivent encore être complétés pour être comparables à ceux découverts par Mint. Cependant, comme le montrent les expériences, appliquer directement Mint à des données discrétisées fines permet d'obtenir de meilleurs résultats que d'appliquer IPD comme étape de prétraitement. Cela peut s'expliquer par le fait que IPD renvoie des limites uniformes ou globales, qui sont moins précises que les limites spécifiquement réglées par Mint pour chaque motif.

Pour résumer, l'algorithme Mint montre diverses très bonnes capacités par rapport à ses concurrents, parmi lesquelles un bon comportement sur des ensembles de données discrétisés fins, une bonne scalabilité, et la production d'un nombre modéré de motifs non redondants avec des limites précises. Cependant, il est encore possible d'améliorer Mint, par exemple en évitant les motifs redondants et en exploitant les régions clairsemées de la même manière que les régions denses.

Bibliography

[AAP00] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 108–118. ACM SIGKDD, 2000.

[AC17] Alexandre Albano and Bogdan Chornomaz. Why concept lattices are large: extremal theory for generators, concepts, and VC-dimension. *International Journal of General Systems*, 46(5), 2017.

[AH14] Charu C. Aggarwal and Jiawei Han, editors. *Frequent Pattern Mining*. Springer, 2014.

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the International Conference on Management of Data*, volume 22, pages 207–216. ACM SIGMOD, 1993.

[And14] Simon Andrews. A partial-closure canonicity test to increase the efficiency of CbO-type algorithms. In *International Conference on Conceptual Structures*, pages 37–50. Springer, 2014.

[Arm74] William Ward Armstrong. Dependency structures of data base relationships. In *Proceedings of the 6th IFIP Congress*, volume 74, pages 580–583, 1974.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, volume 1215, pages 487–499, 1994.

[ATVF12] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. Fast and reliable anomaly detection in categorical data. In *Proceedings of the 21st International Conference on Information and Knowledge Management*, pages 415–424. ACM, 2012.

[AU09] Hiroki Arimura and Takeaki Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proceedings of the International Conference on Data Mining*, pages 1088–1099. SIAM, 2009.

[AY98] Charu C. Aggarwal and Philip S. Yu. A new framework for itemset generation. In *Proceedings of the 17th Symposium on Principles of Database Systems*, pages 18–24. ACM SIGACT-SIGMOD-SIGART, 1998.

[AZ02] Maria-Luiza Antonie and Osmar R. Zaiane. Text document categorization by term association. In *Proceedings of the International Conference on Data Mining*, pages 19–26. IEEE, 2002.

[BAG00] Roberto J. Bayardo, Rakesh Agrawal, and Dimitrios Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2-3):217–240, 2000.

[BBL10] Alexis Bondu, Marc Boullé, and Vincent Lemaire. A non-parametric semi-supervised discretization method. *Knowledge and Information Systems*, 24(1):35–57, 2010.

[BBR03] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

[BCF20] Francesco Bariatti, Peggy Cellier, and Sébastien Ferré. GraphMDL: Graph pattern selection based on minimum description length. In *International Symposium on Intelligent Data Analysis*, pages 54–66. Springer, 2020.

[BCG01] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 443–452. IEEE, 2001.

[BCKN18] Jaume Baixeries, Victor Codocedo, Mehdi Kaytoue, and Amedeo Napoli. Characterizing approximate-matching dependencies in Formal Concept Analysis with pattern structures. *Discrete Applied Mathematics*, 249:18–27, 2018.

[BEJ+13] Aleksey Buzmakov, Elias Egho, Nicolas Jay, Sergei O. Kuznetsov, Amedeo Napoli, and Chedy Raïssi. On projections of sequential pattern structures (with an application on care trajectories). In *Proceedings of the 10th International Conference on Concept Lattices and Their Applications*, volume 1062 of *CEUR Workshop Proceedings*, pages 199–208. CEUR-WS.org, 2013.

[BG09] Mario Boley and Henrik Grosskreutz. Approximating the number of frequent sets in dense data. *Knowledge and Information Systems*, 21(1):65–89, 2009.

[BGG10] Mario Boley, Thomas Gärtner, and Henrik Grosskreutz. Formal concept sampling for counting and threshold-free local pattern mining. In *Proceedings of the International Conference on Data Mining*, pages 177–188. SIAM, 2010.

[BHPW07a] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Efficient closed pattern mining in strongly accessible set systems. In *Proceedings of the 11th European Conference on Principles and Practice of Data Mining and Knowledge Discovery*, pages 382–389. Springer, 2007.

[BHPW07b] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Efficient closed pattern mining in strongly accessible set systems. In *Proceedings of the 5th International Workshop on Mining and Learning with Graphs*, 2007.

[BHPW10] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Listing closed sets of strongly accessible set systems with applications to data mining. *Theoretical Computer Science*, 411(3):691–700, 2010.

[BJ98] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In *Proceedings of the International Conference on Management of Data*, pages 85–93. ACM SIGMOD, 1998.

[BKN14a] Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Characterizing functional dependencies in formal concept analysis with pattern structures. *Annals of Mathematics and Artificial Intelligence*, 72(1-2):129–149, 2014.

[BKN14b] Aleksey Buzmakov, Sergei O. Kuznetsov, and Amedeo Napoli. Scalable estimates of concept stability. In *Proceedings of the 12th International Conference on Formal Concept Analysis*, pages 157–172. Springer, 2014.

[BKN15a] Aleksey Buzmakov, Sergei O. Kuznetsov, and Amedeo Napoli. Fast generation of best interval patterns for nonmonotonic constraints. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 157–172. Springer, 2015.

[BKN15b] Aleksey Buzmakov, Sergei O. Kuznetsov, and Amedeo Napoli. Sofia: how to make FCA polynomial? In *Proceedings of the 4th International Conference on What can FCA do for Artificial Intelligence?*, volume 1430 of *CEUR Workshop Proceedings*, pages 27–34. CEUR-WS.org, 2015.

[BKRK17] Aimene Belfodil, Sergei O. Kuznetsov, Céline Robardet, and Mehdi Kaytoue. Mining convex polygon patterns with formal concept analysis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1425–1432. ijcai.org, 2017.

[BLPG11] Mario Boley, Claudio Lucchese, Daniel Paurat, and Thomas Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *Proceedings of the 17th International Conference on Knowledge discovery and Data Mining*, pages 582–590. ACM SIGKDD, 2011.

[BMG12] Mario Boley, Sandy Moens, and Thomas Gärtner. Linear space direct pattern sampling using coupling from the past. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining*, pages 69–77. ACM, 2012.

[BMS97] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of International Conference on Management of Data*, pages 265–276. ACM SIGMOD, 1997.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the International Conference on Management of Data*, pages 255–264. ACM SIGMOD, 1997.

[Bou06] Marc Boullé. MODL: A Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BTP+00] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. In *ACM SIGKDD Explorations Newsletter*, volume 2. ACM SIGKDD, 2000.

[BV15] Kailash Budhathoki and Jilles Vreeken. The difference and the norm – characterising similarities and differences between databases. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 206–223. Springer, 2015.

[CAP18] Liangzhe Chen, Sorour E Amiri, and B Aditya Prakash. Automatic segmentation of data sequences. In *Proceedings of the 32nd Conference on Artificial Intelligence, the 30th Conference on Innovative Applications of Artificial Intelligence, and the 8th Symposium on Educational Advances in Artificial Intelligence*, pages 2844–2851. AAAI, 2018.

[CBK+17] Victor Codocedo, Guillaume Bosc, Mehdi Kaytoue, Jean-François Boulicaut, and Amedeo Napoli. A proposition for sequence mining using pattern structures. In *Proceedings of the 14th International Conference on Formal Concept Analysis*, pages 106–121. Springer, 2017.

[CG02] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pages 74–86. Springer, 2002.

[CGJ06] Toon Calders, Bart Goethals, and Szymon Jaroszewicz. Mining rank-correlated sets of numerical attributes. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 96–105. ACM SIGKDD, 2006.

[CGSF+13] Loïc Cerf, Dominique Gay, Nazha Selmaoui-Folcher, Bruno Crémilleux, and Jean-François Boulicaut. Parameter-free classification in multi-class imbalanced data sets. *Data & Knowledge Engineering*, 87:109–129, 2013.

[Cha69] Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM (JACM)*, 16(1):145–159, 1969.

[CLN14] Victor Codocedo, Ioanna Lykourentzou, and Amedeo Napoli. A semantic approach to concept lattice-based information retrieval. *Annals of Mathematics and Artificial Intelligence*, 72(1-2):169–195, 2014.

[CN14] Victor Codocedo and Amedeo Napoli. Lattice-based biclustering using partition pattern structures. In *Proceedings of the 21st European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 213–218. IOS Press, 2014.

[Coe03] Frans Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library. `http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS_KDD_DN`, 2003.

[CT91] Thomas M. Cover and Joy A. Thomas. Elements of information theory. *New York*, 68:69–73, 1991.

[CT12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[DB11] Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011.

[DHS19] Dominik Dürrschnabel, Tom Hanika, and Maximilian Stubbemann. Fca2vec: Embedding techniques for formal concept analysis. *arXiv preprint arXiv:1911.11496*, 2019.

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, pages 89–98. ACM SIGKDD, 2003.

[DPD11] Rajashree Dash, Rajib Lochan Paramguru, and Rasmita Dash. Comparative analysis of supervised and unsupervised discretization techniques. *International Journal of Advances in Science and Technology*, 2(3):29–37, 2011.

[DvLDR17] Vladimir Dzyuba, Matthijs van Leeuwen, and Luc De Raedt. Flexible constrained sampling with guarantees for pattern mining. *Data Mining and Knowledge Discovery*, 31(5):1266–1293, 2017.

[Fan49] Robert M. Fano. *The transmission of information: A Statistical Theory of Communication.* Massachusetts Institute of Technology, Research Laboratory of Electronics, 1949.

[FHK06] Johannes Fischer, Volker Heun, and Stefan Kramer. Optimal string mining under frequency constraints. In *Proceedings of the 10th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 139–150. Springer, 2006.

[FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1993.

[FLV+17] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Chi Truong, Ji Zhang, and Hoai Bac Le. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(4), 2017.

[FPS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Mag.*, 17(3):37–54, 1996.

[FS95] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the 2nd European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 1995.

[Fu11] Tak-Chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[FvL20] Micky Faas and Matthijs van Leeuwen. Vouw: Geometric pattern mining using the mdl principle. In *Proceedings of the International Symposium on Intelligent Data Analysis*, pages 158–170. Springer, 2020.

[FVLK+17] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[Gal20] Esther Galbrun. The minimum description length principle for pattern mining: A survey. *arXiv*, 2007.14009, 2020.

[GD86] Jean-Louis Guigues and Vincent Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences humaines*, 95:5–18, 1986.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[GGM04] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Proceedings of the 7th International Conference on Discovery Science*, pages 278–289. Springer, 2004.

[GH06] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[GK01] Bernhard Ganter and Sergei O. Kuznetsov. Pattern structures and their projections. In *Proceedings of the 9th International Conference on Conceptual Structures*, pages 129–142. Springer, 2001.

[GK03] Bernhard Ganter and Sergei O. Kuznetsov. Hypotheses and version spaces. In *Proceedings of the 11th International Conference on Conceptual Structures*, pages 83–95. Springer, 2003.

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[GMMT07] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. Assessing data mining results via swap randomization. *Transactions on Knowledge Discovery from Data (TKDD)*, 1(3):14, 2007.

[Grü07] Peter D. Grünwald. *The Minimum Description Length Principle*. MIT press, 2007.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Springer Berlin Heidelberg, 1999.

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

[HI16] Qiong Hu and Tomasz Imielinski. Alpine: Anytime mining with definite guarantees. *arXiv preprint arXiv:1610.07649*, 2016.

[HI17] Qiong Hu and Tomasz Imielinski. Alpine: Progressive itemset mining with definite guarantees. In *Proceedings of the International Conference on Data Mining*, pages 63–71. SIAM, 2017.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition.* Morgan Kaufmann, 2011.

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

[HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques.* Elsevier, 2011.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *Proceedings of the International Conference on Management of Data*, 29(2):1–12, 2000.

[HS08] Miki Hermann and Barış Sertkaya. On the complexity of computing generators of closed sets. In *Proceedings of the 6th International Conference on Formal Concept Analysis*, pages 158–168. Springer, 2008.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

[HTT09] Tony Hey, Stewart Tansley, and Kristin Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research, October 2009.

[Huf52] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letter*, 31(8):651–666, 2010.

[Jay82] Edwin T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.

[JCN19] Nyoman Juniarta, Miguel Couceiro, and Amedeo Napoli. A unified approach to biclustering based on formal concept analysis and interval pattern structure. In *Proceedings of the International Conference on Discovery Science*, pages 51–60. Springer, 2019.

[JCNR18] Nyoman Juniarta, Miguel Couceiro, Amedeo Napoli, and Chedy Raïssi. Sequence mining within formal concept analysis for analyzing visitor trajectories. In *Proceedings of the 13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP)*, pages 19–24. IEEE, 2018.

[JKK20] Radek Janostik, Jan Konecny, and Petr Krajča. LCM is well implemented CbO: study of LCM from FCA point of view. In *Proceedings of the 15th International Conference on Conceptual Structures*, pages 47–58. Tallinn University of Technology, 2020.

[JMG20] Ian Jeantet, Zoltán Miklós, and David Gross-Amblard. Overlapping hierarchical clustering. In *Proceedings of the 18th International Symposium on Intelligent Data Analysis (IDA)*, volume 12080 of *Lecture Notes in Computer Science 12080*, pages 261–273. Springer, 2020.

[JS04] Szymon Jaroszewicz and Dan A. Simovici. Interestingness of frequent itemsets using Bayesian networks as background knowledge. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining*, pages 178–186. ACM SIGKDD, 2004.

[KH06a] Arno J Knobbe and Eric K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 237–244. ACM SIGKDD, 2006.

[KH06b] Arno J Knobbe and Eric K. Y. Ho. Pattern teams. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pages 577–584. Springer, 2006.

[KKN11a] Mehdi Kaytoue, Sergei O. Kuznetsov, and Amedeo Napoli. Biclustering numerical data in formal concept analysis. In *Proceedings of the International Conference on Formal Concept Analysis*, pages 135–150. Springer, 2011.

[KKN11b] Mehdi Kaytoue, Sergei O. Kuznetsov, and Amedeo Napoli. Revisiting numerical pattern mining with formal concept analysis. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1342–1347, 2011.

[KKND11] Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli, and Sébastien Duplessis. Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences*, 181(10):1989–2001, 2011.

[KKVF14] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the International Conference on Data Mining*, pages 91–99. SIAM, 2014.

[KM07] Petri Kontkanen and Petri Myllymäki. MDL histogram density estimation. In *Artificial Intelligence and Statistics*, pages 219–226, 2007.

[KM18] Sergei O. Kuznetsov and Tatiana Makhalova. On interestingness measures of formal concepts. *Information Sciences*, 442:202–219, 2018.

[KO02] Sergei O. Kuznetsov and Sergei A Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):189–216, 2002.

[Koh95] Ron Kohavi. The power of decision tables. In *Proceedings of the European Conference on Machine Learning*, pages 174–189. Springer, 1995.

[Kol65] Andrei N Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission*, 1(1):1–7, 1965.

[KOR10] Mikhail Klimushkin, Sergei Obiedkov, and Camille Roth. Approaches to the selection of relevant concepts in the case of noisy data. In *Proceedings of the International Conference on Formal Concept Analysis*, pages 255–266. Springer, 2010.

[Kra49] Leon Gordon Kraft. *A device for quantizing, grouping, and coding amplitude-modulated pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

[KS05] Sergei O. Kuznetsov and Mikhail V. Samokhin. Learning closed sets of labeled graphs for chemical applications. In *Proceedings of the International Conference on Inductive Logic Programming*, pages 190–208. Springer, 2005.

[Kuz89] Sergei O Kuznetsov. Interpretation on graphs and complexity characteristics of a search for specific patterns. *Automatic Documentation and Mathematical Linguistics*, 24(1):37–45, 1989.

[Kuz90] Sergei O. Kuznetsov. Stability as an estimate of degree of substantiation of hypotheses derived on the basis of operational similarity. *Nauchn. Tekh. Inf., Ser. 2*, (12):21–29, 1990.

[Kuz91] Sergei O. Kuznetsov. JSM-method as a machine learning method. *Itogi Nauki i Tekhniki, ser. Informatika*, 15:17–50, 1991.

[Kuz93] Sergei O. Kuznetsov. A fast algorithm for computing all intersections of objects from an arbitrary semilattice. *Nauchno-Tekhnicheskaya Informatsiya Seriya 2-Informatsionnye Protsessy i Sistemy*, (1):17–20, 1993.

[Kuz99] Sergei O. Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pages 384–391. Springer, 1999.

[Kuz04] Sergei O. Kuznetsov. Complexity of learning in concept lattices from positive and negative examples. *Discrete Applied Mathematics*, 142(1-3):111–125, 2004.

[Kuz07] Sergei O. Kuznetsov. On stability of a formal concept. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):101–115, 2007.

[KV09] Petr Krajca and Vilem Vychodil. Comparison of data structures for computing formal concepts. In *Proceedings of the 6th International Conference on Modeling Decisions for Artificial Intelligence*, pages 114–125. Springer, 2009.

[KWL+06] Ye Kang, Shanshan Wang, Xiaoyan Liu, Hokyin Lai, Huaiqing Wang, and Baiqi Miao. An ICA-based multivariate discretization algorithm. In *Proceedings of the International Conference on Knowledge Science, Engineering and Management*, pages 556–562. Springer, 2006.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, volume 98, pages 80–86. AAAI Press, 1998.

[LI59] Philip M. Lewis II. Approximating probability distributions to reduce storage requirements. *Information and control*, 2(3):214–225, 1959.

[LKPC14] Hoang Thanh Lam, Julia Kiseleva, Mykola Pechenizkiy, and Toon Calders. Decomposing a sequence into independent subsequences using compression algorithms. In *Proceedings of the Workshop on Interactive Data Exploration and Analytic*, pages 67–75. ACM SIGKDD, 2014.

[LS05] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. In *Formal concept analysis*, pages 180–195. Springer, 2005.

[McM56] Brockway McMillan. Two inequalities implied by unique decipherability. *IRE Transactions on Information Theory*, 2(4):115–116, 1956.

[Mir96] Boris G. Mirkin. Mathematical classification and clustering, 1996.

[Mit79] Tom Michael Mitchell. *Version spaces: an approach to concept learning.* PhD thesis, Stanford University, 1979.

[Mit97] Tom M. Mitchell. Machine learning. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[MK11] Boris G. Mirkin and Andrey V. Kramarenko. Approximate bicluster and tricluster boxes in the analysis of binary data. In *Proceedings of the International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 248–256. Springer, 2011.

[MKC20] Esteban Marquer, Ajinkya Kulkarni, and Miguel Couceiro. Embedding formal contexts using unordered composition. In *Proceedings of the 8th International Workshop "What can FCA do for Artificial Intelligence?" (colocated wit ECAI2020)*, 2020.

[MKN19a] Tatiana Makhalova, Sergei O. Kuznetsov, and Amedeo Napoli. Numerical pattern mining through compression. In *Proceedings of the Data Compression Conference*, pages 112–121. IEEE, 2019.

[MKN19b] Tatiana Makhalova, Sergei O. Kuznetsov, and Amedeo Napoli. On coupling FCA and MDL in pattern mining. In *Proceedings of the 15th International Conference on Formal Concept Analysis*, pages 332–340. Springer, 2019.

[MKN20] Tatiana Makhalova, Sergei O. Kuznetsov, and Amedeo Napoli. Adaptive multi-model approaches to pattern set mining. In *Proceedings of the 9th European Starting AI Researchers' Symposium co-located with 24th European Conference on Artificial Intelligence*, 2020.

[MO04] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[MPM+12] Kartick Chandra Mondal, Nicolas Pasquier, Anirban Mukhopadhyay, Ujjwal Maulik, and Sanghamitra Bandhopadyay. A new approach for association rule mining and bi-clustering using formal concept analysis. In *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 86–101. Springer, 2012.

[MPY05] Sameep Mehta, Srinivasan Parthasarathy, and Hui Yang. Toward unsupervised correlation preserving discretization. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1174–1185, 2005.

[MR14] Oded Z Maimon and Lior Rokach. *Data mining with decision trees: theory and applications*, volume 81. World scientific, 2014.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[MT20] Tatiana Makhalova and Martin Trnecka. From-below boolean matrix factorization algorithm based on MDL. *Advances in Data Analysis and Classification*, pages 1–20, 2020.

[MTU11] Fabian Moerchen, Michael Thies, and Alfred Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowledge and Information Systems*, 29(1):55–80, 2011.

[MTV94] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop. Technical Report WS-94-03*, pages 181–192. Citeseer, 1994.

[MV11] Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining*, pages 51–59. ACM SIGKDD, 2011.

[MV14] Pauli Miettinen and Jilles Vreeken. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(4):1–31, 2014.

[MVT12] Michael Mampaey, Jilles Vreeken, and Nikolaj Tatti. Summarizing data succinctly with the most informative itemsets. *ACM Transactions on Knowledge Discovery from Data*, 6(4):16, 2012.

[MW99] Dimitris Meretakis and Beat Wüthrich. Extending naive bayes classifiers using long itemsets. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 165–174. ACM SIGKDD, 1999.

[NMVB14] Hoang-Vu Nguyen, Emmanuel Müller, Jilles Vreeken, and Klemens Böhm. Unsupervised interaction-preserving discretization of multivariate data. *Data Mining and Knowledge Discovery*, 28(5-6):1366–1397, 2014.

[PBTL99a] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Closed sets based discovery of small covers for association rules. In *BDA'1999 international conference on Advanced Databases*, pages 361–381, 1999.

[PBTL99b] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the International Conference on Database Theory*, pages 398–416. Springer, 1999.

[PBTL99c] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information systems*, 24(1):25–46, 1999.

[PHM00] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery*, volume 4, pages 21–30. ACM SIGMOD, 2000.

[Pia91] Gregory Piatetsky-Shapiro. Knowledge discovery in real databases: A report on the IJCAI-89 workshop. *AI Mag.*, 11(5):68–70, 1991.

[PIKD13] Jonas Poelmans, Dmitry I. Ignatov, Sergei O. Kuznetsov, and Guido Dedene. Formal concept analysis in knowledge processing: A survey on applications. *Expert Systems with Applications*, 40(16):6538–6560, 2013.

[PKID13a] Jonas Poelmans, Sergei O Kuznetsov, Dmitry I Ignatov, and Guido Dedene. Formal concept analysis in knowledge processing: A survey on models and techniques. *Expert systems with applications*, 40(16):6601–6623, 2013.

[PKID13b] Jonas Poelmans, Sergei O. Kuznetsov, Dmitry I. Ignatov, and Guido Dedene. Formal concept analysis in knowledge processing: A survey on models and techniques. *Expert Systems with Applications*, 40(16):6601–6623, 2013.

[PMS03] Dmitry Pavlov, Heikki Mannila, and Padhraic Smyth. Beyond independence: Probabilistic models for query approximation on binary transaction data. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1409–1421, 2003.

[PRB05] Ruggero G Pensa, Céline Robardet, and Jean-François Boulicaut. A bi-clustering framework for categorical data. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pages 643–650. Springer, 2005.

[PS91] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. *Knowledge Discovery in Databases*, pages 229–238, 1991.

[PTB+05] Nicolas Pasquier, Rafik Taouil, Yves Bastide, Gerd Stumme, and Lotfi Lakhal. Generating a Condensed Representation for Association Rules. *Journal of Intelligent Information Systems*, 24(1):29–60, 2005.

[PvL20] Hugo M. Proença and Matthijs van Leeuwen. Interpretable multiclass classification by mdl-based rule lists. *Information Sciences*, 512:1372–1393, 2020.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Ris83] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, pages 416–431, 1983.

[RKF12] Saif Ur Rehman, Asmat Ullah Khan, and Simon Fong. Graph mining: A survey of graph mining techniques. In *Proceedings of the 7th International Conference on Digital Information Management*, pages 88–92. IEEE, 2012.

[RSY92] Jorma Rissanen, Terry P. Speed, and Bin Yu. Density estimation by stochastic complexity. *IEEE Transactions on Information Theory*, 38(2):315–323, 1992.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the International Conference on Management of Data*, pages 1–12. ACM SIGMOD, 1996.

[SGIK14] Fedor Strok, Boris Galitsky, Dmitry Ilvovsky, and Sergei O. Kuznetsov. Pattern structure projections for learning discourse structures. In *Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 254–260. Springer, 2014.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[SHS+00] Pradeep Shenoy, Jayant R. Haritsa, S. Sudarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. Turbo-charging vertical mining of large databases. *ACM Sigmod Record*, 29(2):22–33, 2000.

[SK01] Masakazu Seno and George Karypis. LPMiner: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Proceedings of the International Conference on Data Mining*, pages 505–512. IEEE, 2001.

[SK11] Arno Siebes and René Kersten. A structure function for transaction data. In *Proceedings of the International Conference on Data Mining*, pages 558–569. SIAM, 2011.

[sli] SLIM: Directly mining descriptive patterns (slides from SDM 2012). `https://eda.mmci.uni-saarland.de/pres/sdm12-slim-pres.pdf`. Accessed: 2019-11-01.

[Sol64] Ray J. Solomonoff. A formal theory of inductive inference. Part I. *Information and control*, 7(1):1–22, 1964.

[SR14] Arnaud Soulet and François Rioult. Efficiently depth-first minimal pattern mining. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 28–39. Springer, 2014.

[STB+02] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with titanic. *Data & knowledge engineering*, 42(2):189–222, 2002.

[SV12] Koen Smets and Jilles Vreeken. Slim: Directly mining descriptive patterns. In *Proceedings of the International Conference on Data Mining*, pages 236–247. SIAM, 2012.

[SV19] Divyam Saran and Jilles Vreeken. Summarizing dynamic graphs using MDL. 2019.

[SVN+14] Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, Robert Godin, Alix Boc, and Vladimir Makarenkov. A fast compound algorithm for mining generators, closed itemsets, and computing links between equivalence classes. *Annals of Mathematics and Artificial Intelligence*, 70(1-2):81–105, 2014.

[SVNG09] Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, and Robert Godin. Efficient vertical mining of frequent closures and generators. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 393–404. Springer, 2009.

[Tat13] Nikolaj Tatti. Itemsets for real-valued datasets. In *Proceedings of the 13th International Conference on Data Mining*, pages 717–726. IEEE, 2013.

[TV08] Nikolaj Tatti and Jilles Vreeken. Finding good itemsets by packing data. In *Proceedings of the 8th International Conference on Data Mining*, pages 588–597. IEEE, 2008.

[TV12a] Nikolaj Tatti and Jilles Vreeken. Discovering descriptive tile trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 9–24. Springer, 2012.

[TV12b] Nikolaj Tatti and Jilles Vreeken. Discovering Descriptive Tile Trees – By Mining Optimal Geometric Subtiles. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science 7523, pages 9–24. Springer, 2012.

[TV12c] Nikolaj Tatti and Jilles Vreeken. The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 462–470, 2012.

[UAUA03] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. LCM: An efficient algorithm for enumerating frequent closed item sets. In *Proceedings of the ICDM Workshop on Frequent Itemset Mining Implementations*, volume 90. Citeseer, 2003.

[UAUA04] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Proceedings of the 7th International Conference on Discovery Science*, pages 16–31. Springer, 2004.

[UKA04] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, volume 126, 2004.

[UKA05] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 77–86, 2005.

[VC71] Vladimir Naumovich Vapnik and Aleksei Yakovlevich Chervonenkis. On uniform convergence of the frequencies of events to their probabilities. *Teoriya Veroyatnostei i ee Primeneniya*, 16(2):264–279, 1971.

[VC15] Vladimir N. Vapnik and Alexey Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[vCDB17] Toon van Craenendonck, Sebastijan Dumancic, and Hendrik Blockeel. COBRA: A fast and simple method for active clustering with pairwise constraints. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2871–2877, 2017.

[VL97] Paul Vitanyi and Ming Li. *An introduction to Kolmogorov complexity and its applications*, volume 34. Springer Heidelberg, 1997.

[VMZ06] Adriano Veloso, Wagner Meira, and Mohammed J. Zaki. Lazy associative classification. In *Proceedings of the 6th International Conference on Data Mining*, pages 645–654. IEEE, 2006.

[VT14] Jilles Vreeken and Nikolaj Tatti. Interesting patterns. In Charu C. Aggarwal and Jiawei Han, editors, *Frequent Pattern Mining*, pages 105–134. Springer, 2014.

[VVLS11] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[WDKG14] Jouke Witteveen, Wouter Duivesteijn, Arno Knobbe, and Peter Grünwald. Realkrimp – finding hyperintervals that compress with MDL for real-valued data. In *Proceedings of the International Symposium on Intelligent Data Analysis*, pages 368–379. Springer, 2014.

[Web10] Geoffrey I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–20, 2010.

[Wit12] Jouke Witteveen. Mining hyperintervals – getting to grips with real-valued data, 2012.

[WK04] Jianyong Wang and George Karypis. BAMBOO: Accelerating closed itemset mining by deeply pushing the length-decreasing support constraint. In *Proceedings of the International Conference on Data Mining*, pages 432–436. SIAM, 2004.

[WP06] Chao Wang and Srinivasan Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 730–735. ACM SIGKDD, 2006.

[Zak80] Shmuel Zaks. Lexicographic generation of ordered trees. *Theoretical Computer Science*, 10(1):63–82, 1980.

[Zak00a] Mohammed J. Zaki. Generating non-redundant association rules. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 34–43. ACM SIGKDD, 2000.

[Zak00b] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[ZH99] Mohammed J. Zaki and Ching-Jui Hsiao. CHARM: An efficient algorithm for closed association rule mining. Technical report, Citeseer, 1999.

[ZH02] Mohammed J. Zaki and Ching-Jui Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the International Conference on Data Mining*, pages 457–473. SIAM, 2002.

[ZN14] Albrecht Zimmermann and Siegfried Nijssen. Supervised pattern mining and applications to classification. In Charu C. Aggarwal and Jiawei Han, editors, *Frequent Pattern Mining*, pages 425–442. Springer, 2014.

[ZO98] Mohammed J. Zaki and Mitsunori Ogihara. Theoretical foundations of association rules. In *Proceedings of the 3rd Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 71–78. ACM SIGMOD, 1998.

Résumé

Dans cette thèse, nous étudions différents aspects de l'exploration ou fouille de motifs dans des jeux de données tabulaires (binaires et numériques). L'objectif de l'exploration de motifs est de découvrir un petit ensemble de motifs non redondants qui peuvent recouvrir presque entièrement un ensemble de données et être interprétés comme des unités de connaissances significatives et utiles. Nous nous concentrons sur certaines questions clés telles que (i) la définition formelle de l'intérêt des motifs, (ii) la minimisation de l'explosion combinatoire des motifs, (iii) la définition de mesures pour évaluer les performances des méthodes d'exploration de motifs, et (iv) le rapprochement entre l'intérêt et la qualité des ensembles de motifs. De plus, nous allons au-delà des investigations standards sur l'exploration de motifs en faisant émerger et en étudiant la structure intrinsèque sous-jacente à un jeu de données tabulaire. Les principales contributions de ce travail de recherche sont à la fois théoriques, conceptuelles et pratiques. En ce qui concerne la nouveauté théorique, nous proposons une structure dite "de niveaux de fermetures" et l'algorithme GDPM qui la calcule. La structure de niveaux de fermetures nous permet d'estimer à la fois la complexité des données et celle des motifs. En outre et en pratique, cette structure peut être utilisée pour représenter la topologie des données par rapport à une mesure d'intérêt. Du point de vue conceptuel, la structure de niveaux de fermeture autorise un analyste à comprendre la configuration intrinsèque des données avant de sélectionner une mesure d'intérêt plutôt que de comprendre les données au moyen d'une mesure d'intérêt choisie arbitrairement. Dans cette thèse, nous discutons également de la différence entre l'intérêt et la qualité des ensembles de motifs. Nous proposons d'adopter les bonnes pratiques de l'apprentissage supervisé et de les adapter à la fouille de motifs. Ainsi, nous avons développé un algorithme d'exploration d'ensembles de motifs appelé KeepItSimple, qui met en relation l'intérêt et la qualité des ensembles de motifs. En pratique, KeepItSimple permet de retrouver de façon efficace un ensemble de motifs intéressants sans craindre d'explosion combinatoire. De plus, nous proposons un algorithme glouton d'énumération de motifs susceptibles d'intérêt qui remplace les méthodes classiques d'énumération de motifs fermés fréquents lorsque les motifs sont trop nombreux. Enfin une dernière contribution porte sur le développement d'un algorithme qui s'appuie sur le principe MDL appelé Mint qui a pour objectif d'extraire des ensembles de motifs dans les données numériques. L'algorithme Mint repose sur des bases théoriques solides tout en ayant l'objectif pratique de retourner un ensemble concis de motifs numériques qui sont non redondants et informatifs. Les expérimentations montrent que Mint a un très bon comportement en fouille de données numériques et surpasse généralement ses concurrents en efficacité et qualité des motifs retournés.

Mots-clés: Fouille de motifs · Intérêt des motif · Principe de Longueur de Description Minimale · Motifs fermés · Classes d'équivalence · Complexité des données · Structure de niveaux de fermetures · Explosion des motifs · Évaluation des motifs · Analyse de concept formelle · Structures de modèle d'intervalle · Données binaires · Données numériques

Abstract

In this thesis we study different aspects of pattern mining in binary and numerical tabular datasets. The objective of pattern mining is to discover a small set of non-redundant patterns that may cover entirely a given dataset and be interpreted as useful and significant knowledge units. We focus on some key issues such as (i) formal definition of pattern interestingness, (ii) the minimization of pattern explosion, (iii) measure for evaluating the performance of pattern mining, and (iv) the discrepancy between interestingness and quality of the discovered pattern sets. Moreover, we go beyond the typical perspectives of pattern mining and investigate the intrinsic structure underlying a tabular dataset. The main contributions of this research work are theoretical, conceptual, and practical. Regarding the theoretical novelty, we propose a so-called closure structure and the GDPM algorithm for its computing. The closure structure allows us to estimate both the data and pattern complexity. Furthermore, practically the closure structure may be used to represent the data topology w.r.t. an interestingness measure. Conceptually, the closure structure allows an analyst to understand the intrinsic data configuration before selecting any interestingness measure rather than to understand the data by means of an arbitrarily selected interestingness measure. In this research work, we also discuss the difference between interestingness and quality of pattern sets. We propose to adopt the best practices of supervised learning in pattern mining. Based on that, we developed an algorithm for itemset mining, called KeepItSimple, which relates interestingness and the quality of pattern sets. In practice, KeepItSimple allows us to efficiently mine a set of interesting and good-quality patterns without any pattern explosion. In addition, we propose an algorithm for a greedy enumeration of likely-occurring itemsets that can be used when frequent closed itemset miners return too many itemsets. The last practical contribution consists in developing an MDL-based algorithm called Mint for mining pattern sets in numerical data. The Mint algorithm relies on a strong theoretical foundation and at the same time has a practical objective in returning a small set of numerical, non-redundant, and informative patterns. The experiments show that Mint has very good behavior in practice and usually outperforms its competitors.

Keywords: Pattern Set Mining · Pattern interestingness · MDL · Minimum Description Length principle · Closed patterns · Equivalence classes · Data complexity · Closure structure · Pattern explosion · Pattern evaluation · Formal Concept Analysis · Interval Pattern Structures · Binary data · Numerical data