KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING INFORMATICA
Celestijnenlaan 200 A — B-3001 Leuven

MINING SETS OF PATTERNS

Promotor :
Prof. Dr. L. DE RAEDT

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Albrecht ZIMMERMANN

May 2009

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING INFORMATICA
Celestijnenlaan 200 A — B-3001 Leuven

MINING SETS OF PATTERNS

Jury :
Prof. Dr. ir. J. Berlamont, voorzitter
Prof. Dr. L. De Raedt, promotor
Prof. Dr. ir. H. Blockeel
Prof. Dr. B. Berendt
Prof. Dr. B. Goethals (Universiteit Antwerpen, Belgium)
Prof. Dr. A. Siebes (Universiteit Utrecht, Netherlands)

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Albrecht ZIMMERMANN

U.D.C. 681.3*I26

May 2009

ISBN 978-94-6018-077-4
D/2009/7515/60

Abstract

Local pattern mining is an integral part of a variety of approaches in data mining and machine learning. Especially in data mining, a lot of research exists on how to mine local patterns efficiently from large data bases. As a side-effect, the resulting pattern collections are far too big to be useful. In machine learning, on the other hand, local patterns are ingredients to more complex models used for classification, or clustering, for instance.

In both cases, efficiently and effectively assembling those local patterns into *sets of patterns* is an important task. In this thesis, we present a in-depth discussion of various aspects of *pattern set mining*.

The thesis has three main contributions. The first one is the introduction of a formal framework for pattern set mining as a distinct task. To the best of our knowledge, this is the first time that such a formal definition has been given. By identifying types of pattern sets, properties of those types, and developing constraints based on these properties, the framework enables the principled discussion of existing and future pattern set mining techniques.

As a direct benefit, in a second step, we leverage the framework for the discussion of existing approaches from data mining and machine learning. We identify the two main approaches to pattern set mining, post-processing and iterative mining, and characterize existing heuristic techniques. Following our discussion, we suggest an exhaustive alternative to existing heuristic data mining techniques, as well as possibilities for the upgrade of those techniques. Additionally, we inspect the local pattern mining approaches to which most pattern set mining techniques from machine learning are coupled and argue that they can be replaced by different methods.

Finally, to validate our theoretical findings, we perform a number of experimental evaluations of existing and proposed systems. We show that an exhaustive post-processing method does indeed allows us to perform constraint-based pattern set mining. We also evaluate the effect of heuristic parameters, such as orders and quality measures, for heuristic post-processing. Additionally, we explore the application of exhaustive local pattern mining in iterative pattern set mining, showing both its applicability. A surprising finding in this last setting is that exhaustive methods, guided by iterative mining, are not more computationally expensive than their heuristic counterparts.

Acknowledgments

Whenever I was thinking about a potential academic career several years ago, there was one statement that I repeated time and again to friends and family: "I will not try and get a Ph.D.!". This was before I enjoyed the research I had to do for my diploma thesis so much that I wanted more. This smaller research project did not prepare me for the rougher patches of Ph.D. work though, and I would not have been able to complete my work and write this thesis without the support of a lot of people. First and foremost, I would like to thank my supervisor Luc De Raedt. From waking my interest in machine learning and data mining in the first place through his lectures, all the way to putting me in a position to attain my Ph.D. successfully, he has been a source of inspiration, motivation and insight. He constantly encouraged me to look outside the immediate area I was working in, to focus on questions rather than answers, and whenever I felt that I was at a dead end, he showed me that it was in fact a fork in the road. He also introduced me to other excellent researchers from around the world, and finally, he exhibited astonishing patience with me.

My colleagues in Freiburg and Leuven had also a significant effect on my work and my life. Specifically, I would like to thank Björn Bringmann whose working style and temperament complement mine so well that we managed to succeed with a number of projects, from papers to presentations to workshop organizations – usually at the latest hour. My other office mates were also invaluable: Siegfried Nijssen who seems to know the answer to any question and if not, will know where to look (prompting Björn to dub him "The Oracle"), and Anton and Tias who gave me a fresh perspective on the field and continuously reminded me not to take myself too serious. It would take too much time and space to comment on every member of the groups in Freiburg and Leuven, even though they all merit it, so I will only point to a few. Kurt Driessens and Kristian Kersting taught me that a presentation does not have to be dry to be informative, Joaquin Vanschoren and Fabrizio Costa always allowed me to distract them with meta-discussions, Niels Landwehr proved to be "The Oracle Mk.II" and was very helpful in the final phase of my Ph.D., as was Daan Fierens – and Elisa Fromont introduced me to Poker and correctly predicted that I would not be very good at it once real money got involved.

Maybe the most surprising characteristic of our field is its flat hierarchies and I had the pleasure to meet many accomplished researchers and draw inspiration from them. I am lucky to have four of them, Bettina Berendt, Hendrik Bloc-

keel, Bart Goethals, and Arno Siebes, serving on my Ph.D. committee. Their comments on the initial version of this thesis greatly helped me to put things into perspective and make the final text far better than it initially was. I would also like to thank Johannes Fürnkranz for making me understand quite a few things about pattern mining and rule learning, and, once again, for his patience. I also thank Prof. Jean Berlamont for chairing my defense.

I gratefully acknowledge the financial support received for the work performed during this thesis from the following organizations: European Union IST-2000-26469 project "consortium on discovering knowledge with Inductive Queries (CINQ)", European Union FP6-516169 project on "Inductive Queries for Mining Patterns and Models", and the BOF-DOC Grant of the KU Leuven.

Last but not least, I want to thank my family and friends for all the support they gave me. My parents were always there when I needed to step back and recharge after stressful times and didn't mind me working on Christmas morning too much. My brother never understood that his achievements are far more impressive than mine and encouraged me whenever its was necessary. Wolfgang and Veeck (and many others) were a source of strength and gave me examples to aspire to. Marie, finally, made working weekends downright enjoyable and always motivated me to work for "the greater good".

Contents

List of Figures

List of Algorithms

List of Tables

> The History of every major Galactic Civilization tends to pass through three distinct and recognizable phases, those of Survival, Inquiry and Sophistication, otherwise known as the How, Why and Where phases. For instance, the first phase is characterized by the question "How can we eat?" the second by the question "Why do we eat?" and the third by the question "Where shall we have lunch?"
>
> Douglas Adams

Introduction

The knowledge discovery in databases (KDD) process is described by (Fayyad et al. 1996) in the following way:

> KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

In discussing data mining, it is important to keep this definition in mind, since it still underlies the self-perception of most of the KDD (and to the related to this data mining) community. The five traits which are used to characterize the process influence both the building blocks of knowledge that are derived – patterns – and the techniques and criteria that are used to find them. Particularly:

- *Nontrivial* refers to the fact that simple statistics of the data, such as modes or means of variable values, for instance, or the strength of correlation between any two variables, is not the goal of KDD. In this regard, KDD is different from statistics, even though it uses statistical methods.
- *Valid*, according to Fayyad *et al.*'s interpretation, means that patterns will hold in so far unseen data and are not specific to the data used for deriving them. Statistical methods help in achieving this goal.
- Patterns should be *novel* to "the system and preferably [also] to the user", which implies a certain degree of unexpectedness given current knowledge.
- *Usefulness* denotes that the user or the task for which KDD is performed benefit from the found patterns.
- Finally, *understandability* is what is needed for obtaining actual *knowledge* since knowledge implies understanding of phenomena.

The only of those terms that is arguably uncontested is the first one, nontriviality in the sense in which it is defined by the paper, since the distinction is made explicit. Validity is already a more vague concept due to the fact that it can almost by definition only be ascertained by the application of found patterns to large amounts of unseen data. Similarly, novelty is connected to expectations of the user, which he might not necessarily be aware of. The concepts of usefulness

and understandability, finally, depend strongly on the KDD task at hand and on the users who initiate it.

Nevertheless, Fayyad *et al.* suggest a notion of *interestingness* that combines the four different characteristics of pattern. Data mining is the step in KDD in which algorithms are applied to the data to enumerate patterns and determine which of those can be considered *interesting*, and if they are sufficiently interesting, knowledge. It is preceded by data collection and preparation, and the mined patterns are interpreted to obtain knowledge. Interestingness can in this view be determined by quantifying the validity, novelty, usefulness, and understandability of patterns and setting thresholds on the resulting values.

While we view data mining in much the same way, that is that it should be considered as mining interesting patterns, this is not necessarily the view of the entire community. Data mining is also concerned with efficiently handling large amounts of low-level data and for parts of the community, the focus is therefore on database management. Applications as a driving force for certain approaches to data mining are another angle. The influence of statistics, finally, can hardly be overstated and there is on-going research into the use and meaning of statistical measures in data mining (Wu et al. 2007).

We chose the quote (Adams 1979) at the beginning of this chapter as a way of describing data mining. While it is not a major Galactic Civilization per se, it is a very significant movement in KDD and it hungers as well – for knowledge. The "how" question has been solved in the field of data mining and especially local pattern mining quite a while ago; in fact, it was essentially settled in 1995 with the publication of the Apriori algorithm (Agrawal and Srikant 1994). Working on itemsets and efficiently enumerating all frequent patterns and confident association rules, it laid the foundation for local pattern mining and spawned a rich culture. The choice of measures and sensible thresholds for interestingness is far from straight-forward, and has been the subject of extensive research in the data mining community (see (McGarry 2005) and (Silberschatz and Tuzhilin 1996) for an overview and discussion of interestingness measures). Combining measures with thresholds leads to the formation of *constraints* which have to be satisfied for a pattern to be considered *interesting*.

The initial constraint, minimum support, and the pruning it enabled to efficiently traverse lattices and prune away uninteresting pattern subspaces is based on the anti-monotonicity property, answering the "why" question, albeit slightly different from Adams' formulation. The question in the case of pattern mining was rather "Why does it work?". Its answer led to the identification of further constraints and properties that have been used to propose efficient algorithms for performing the constraining search ((Han et al. 2007) discusses constraint-classes and algorithmic solutions). It also allowed researchers to extend the mechanisms towards other pattern languages, such as sequences, trees and graphs, and produce yet more (and more complex) patterns.

We have used the term "patterns" repeatedly so far without stating explicitly what they are. According to the work of Fayyad *et al.*,

> [a] pattern is an expression in some language describing a subset

> of the data or a model applicable to the subset.

It is at this point that *understandability*, comes somewhat into conflict with the pattern definition. The authors give a *neural network*, an artificial approximation of the network of synapses which make up the human brain, as an example of a model that is hard to understand. Yet in the same work, discussing data mining methods, neural network formation is mentioned as a data mining technique for non-linear prediction. Further illumination is given by a note appended to the text, which reads:

> Throughout this article, we use the term pattern to designate a pattern found in data. We also refer to models. One can think of patterns as components of models, for example, a particular rule in a classification model or a linear component in a regression model.

It is this view of patterns, as local phenomena, which can be components of models of the data, that we adopt in this thesis. Examples for such patterns are, for instance, frequent itemsets that describe supermarket items that are often bought together, or molecular fragments that occur more often in carnogenic substances than in harmless ones. This means that we do not discuss the induction of models of the entire data, such as Bayesian networks, neural networks, or support vector machines, to name a few. Finally, Fayyad *et al.* state that machine learning techniques can be used in the data mining step of the KDD process. Machine learning is the field of research concerned with making computers *learn* (cf. (Mitchell 1997)):

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

An important subject of machine learning is *model induction*, both for *predictive* tasks such as classification and regression, and for *descriptive* ones, such as clustering. Additionally, when the goal is giving computers the ability to learn, the (automated) KDD process is arguably an important means for achieving this goal. So, even though the similarities between data mining and machine learning are quite tangible, and it is difficult to clearly distinguish the two, the respective research communities perceive themselves as distinct. As an effect of this, similar solutions to the same problem have been proposed in each field, but often failed to stimulate further developments in the other one. It is not the main goal of our work to bridge this gap. But by pointing out similarities between the data mining community, concerned with mining patterns, and the machine learning community, concerned with learning rules and inducing decision trees, we intend to at least narrow it. When we refer to machine learning and data mining in the rest of this thesis, we therefore refer to the research communities and their respective literature on rule learning and pattern mining, and *not* to two factually distinct areas.

Coming back to Adams, "How?" is no longer a problem; instead, nowadays it is rather easy to mine very large amounts of patterns satisfying various constraints efficiently. Too many, as we will discuss below. The inquiry phase is also slowing down to a certain degree, most of the underlying mechanisms are well-understood and have given rise to more sophisticated miners (Bucila et al. 2002) and concepts (Kifer et al. 2003). Interestingness in terms of validity and novelty of found patterns has been studied extensively in the data mining literature (see the survey in (McGarry 2005)), but the problems of understandability and usefulness less so. In the case of individual patterns, understandability can be expressed as simplicity, for instance by restricting the number of items in an itemset. Once patterns occur together in the form of a set however, this becomes more problematic. So pattern mining enters the sophistication phase: not blindly accepting large unstructured sets of patterns but instead selecting which ones to digest.

Data mining in the sense of pattern mining is a means to an end, used in techniques that address a variety of KDD tasks. Among those are *descriptive* tasks, for instance *association rule mining* that can discover associations between purchase decisions of customers. Especially in the case of a large assortment of items that can be bought, in combination with a large customer base, for instance in a web-shop, many of those rules will be discovered, potentially millions. To use this discovered knowledge for the optimization of the organization of the shop, it is necessary to identify which rules are really useful. Especially in the case of a large number of valid and novel rules, however, it is not straight-forward to identify which of those to act on.

The main reason for overly large pattern sets lies in the fact that in traditional data mining patterns are considered individually, without recourse to any background knowledge or to other patterns present in the set of which they are a member. This is largely a side effect of the pruning strategies used, which allow information gained by evaluating constraints on individual patterns to be turned into pruning decisions on entire pattern subspaces.

The foundations for the selection of relevant subsets of patterns were laid quite early when the machine learning concept of *version spaces* (Mitchell 1997) was adapted towards the solution set of a conjunction of anti-monotone and monotone constraints. The derived *borders* can then be used to characterize the solution set (Mannila and Toivonen 1997). Far less compact but more informative while at the same time removing trivial redundancies (on the data they were mined from) are collections of *closed*, *free*, and *non-derivable* patterns (Taouil et al. 2000, Boulicaut and Jeudy 2001, Calders and Goethals 2002).
The strong advantage of the properties used for constructing those so-called *condensed* representations is that all of them can be checked during the mining process itself. This hints at another characteristic though: all these properties are still very much properties of individual patterns. Their scope is somewhat wider in that their presence (and the absence of patterns that are syntactically related to them) implies information about the characteristics of their syntactic relatives. Now our claim that the redundancies removed should be considered *trivial* becomes clearer as well since only those patterns are affected for which

syntactical relatedness exists. The resulting sets are an approximation of a representative subset of the entire pattern set but their large size is still counterproductive in terms of understandability.

Furthermore, there can still be semantic redundancies between unrelated patterns and removing or highlighting those redundancies can be expected to strongly benefit both human end-users and systems processing the mined patterns further, increasing usefulness. This is potentially treacherous ground to tread since the decision what makes up semantic redundancy is influenced by the context of the task for which patterns are mined as well as by user assumptions. Nevertheless, the goal is to do this in a principled way that allows discussing properties of techniques and giving guarantees regarding the solutions found. In short, it requires a notion of *mining sets of patterns*; (Han et al. 2007) identifies this as an important future research direction.. Moreover, there can be additional relationships between the patterns that one wishes to constrain, a task that becomes far easier if the way to mine a particular set of patterns is well understood. In a sense, this approach adds an additional layer to the data mining step in KDD, a view that is also adopted in the formulation of the LeGo framework (Knobbe et al. 2008).

So while the "Why?" question of local pattern mining has been answered, the data mining community has given little thought to the "Why?" question of *pattern set mining*. The main contribution of these thesis is that we supply a framework in which this question can be asked in the first place, and that we discuss some possible solutions to this question. This also leads directly into the formulation of a generic exhaustive algorithm for constraint-based pattern set mining, the first of its kind to the best of our knowledge.

State of the Art

In *predictive* mining, such as *concept learning*, where the goal is to build a working model that can predict labels for unseen data, too many patterns can be a negative phenomenon as well. If patterns are used indirectly, as descriptors of data, it is very important to identify the relevant patterns that give learning algorithms a good chance of modeling the underlying concepts. If patterns are used directly, as is the case in predictive rule mining, redundancies among rules and statistically weak rules can distort predictions. Furthermore, if the user attempts to analyze the learned classifier, too much complexity will be detrimental.

Machine learning focussed early on concept learning from labeled data, leading to classification, or the assignment of unlabeled data to clusters, aiming at modeling unknown classes. There exist methods for inducing meaningful rules in the presence of other rules, basically building a set of predictive patterns. This is an example for the overlap between machine learning and data mining since mining those rules is data mining according to Fayyad *et al.*. The effect of this is that pattern set mining *has* been practiced in a variety of ways for more than a decade by now in the field of machine learning. Since the

data used for experimental evaluation often were small, the efficient enumeration of *local patterns* was not that important, the quality of final pattern sets was, however. The algorithms developed for pattern set selection were therefore heuristic and usually integrated with the local pattern mining step (Clark and Niblett 1989, Cohen 1995, Quinlan 1993), giving the appearance of a complete system, instead of two decomposable steps. To increase the probability of learning a high quality classifier, machine learning also employs methods for feature selection that improve the description of data. This is very much like selecting a subset of patterns mined for descriptive purposes by what would typically be considered a data mining process, as (Knobbe and Ho 2006b) and (Cheng et al. 2007) point out. Since machine learning approaches are mostly agnostic as to the origins of the features, this connection has not been made often otherwise.

Additionally to this relative obscurity of the pattern set mining techniques existing in machine learning, there is another reason that these approaches have, until recently, found little entry into the field of data mining. The reason is simply that they come from the field of machine learning. Beginning from different origins, the two fields have co-existed for quite a while, coming up with different jargon, claiming different focus and often developing similar ideas at the same time. Data mining originally focussed mainly on *large* data bases and how to efficiently mine local patterns from them even though Fayyad *et al.* admit the use of machine learning techniques. A strong focus therefore lay on integration with data base management systems, efficient algorithms scaling well when the amount of data increased, and a certain connection to the business community. Additionally, aiming to use the modeling approaches of machine learning in the business world involves dealing efficiently with large data sources. Since machine learning systems seem to be monolithic, there have been few attempts to decompose them and replace the local pattern mining module with more efficient solutions developed in data mining. On the other hand have data mining approaches been used for concept learning purposes, as in the CBA and CMAR systems (Liu et al. 1998, Li et al. 2001). To build the final model, slightly different methods for assembling a set of predictive patterns have been used, without making the pattern set mining aspect explicit.

The result has been a gradual merging of data mining and machine learning, resulting in works that point out the similarities. What it has not yet led to is a consistent terminology and especially the application of pattern set mining techniques from machine learning in data mining. It is precisely this topic that we investigate in this thesis. As we have explained above, we propose a framework for studying pattern set mining, analyze algorithms and identify properties that allow us to design new ones. The second contribution of our work lies in the combination of heretofore unexplored algorithmic solutions to pattern set mining, mainly by decomposing and applying machine learning techniques for pattern set mining. Before we outline the organization of the thesis, we discuss along which dimensions pattern set mining can be described. Since we already made the distinction between machine learning and data mining, a natural starting point are the KDD tasks in which pattern set mining is applied.

Use of the Mined Patterns

Local pattern mining does not exist as an end in itself. The mining operation usually serves a purpose and this purpose influences the kind of patterns mined, the constraints that have to be satisfied and, of course, also the desired characteristics of pattern sets.

On the one hand, there are those tasks mostly concerned with finding *descriptions* of the data, that is, patterns that can be interpreted to understand the composition of certain subsets in the data. An example case would be *subgroup discovery* (Wrobel 1997), in which distinct subsets of the data have been identified and the practitioner seeks a compact way of describing the elements of each of these subsets. Ideally, not too many of the local descriptions should refer to the same subset. Additionally, a measure of quality of individual patterns is desired, and a way of capturing the relation between them. Most importantly, however, the sets should be easily interpretable, a requirement, which is helped by the relationship between patterns but also one that very much requires these sets to be of small cardinality. Thus, usefulness can be approximated by understandability in these tasks. *Clustering* (Perkowitz and Etzioni 1999), *subgroup discovery*, and *association rule discovery* (Agrawal and Srikant 1994) all fit this template of identifying subsets in the data and finding descriptions for each of them, and pattern set mining is needed to assemble interpretable sets.

On the other hand, *predictive* models can be built from patterns, as we sketched above. The model should, of course, be accurate but it should be noted that there are ways of achieving this even without very accurate local patterns, for instance, by trading off individual errors against each other. This requires, however, that such errors are not amplifying each other, so patterns should not be too similar semantically. Additionally, it is of interest that certain areas of the data (for instance, the majority class) are not covered much more extensively than others.

Even though these applications of pattern set mining serve different purposes, the existing similarities show that a general form of pattern set mining could be found that captures those similarities. There are also certain desired characteristics of pattern sets that emerge: small size of the sets, little redundancy, and often a need to somehow relate patterns to each other.

Different Types of Sets

A way of relating patterns to each other lies in moving from unordered sets (the typical result of local pattern mining in the data mining sense) to ordered or tree sets of patterns. Unordered sets have the advantage of extreme flexibility since there is no predefined order in which patterns should be considered. This means both that algorithmically different techniques can be employed to use those patterns (for instance, in *concept learning*), and that humans can pick and choose which patterns to look at. The potential downside of this is that relations between patterns are implicit, which means it is not immediately clear which patterns influence other patterns' presence with their own. Therefore,

one has to consider the *full* set to get a sense of how patterns relate to each other, making low cardinality even more important. To achieve this, it is often necessary to constrain the redundancy among patterns explicitly, giving rise to potentially arbitrary thresholds, as we will illustrate in Section 3. Additionally, from a practical point of view, unordered sets often require computationally more expensive methods during mining.

Ordered sets give up this flexibility by turning the set into a list where each pattern has to be considered in turn. Therefore, patterns only have meaning in the context of their predecessor, making it easier to interpret them since the user has only to refer back to earlier patterns. This leads to approaches that allow to control redundancy implicitly, decreasing the influence of user intuition on the mining process. On the other hand, not all techniques for processing the final pattern set can be used with ordered sets. Also, the relation that seems to exist according to the order might not be that strong in reality, with a pattern following another one even though both describe very different areas in the data. A rather straight-forward example to ordered set mining we will consider in this work is that of top-k mining, where each pattern is quantified by a scoring function. Its position in the set is thus influenced by the performance of preceding patterns.

Tree sets, finally, relate patterns to each other in a more complex manner, allowing more than one successor to each pattern. This reflects the fact that the relations between patterns are often not such simple affairs as ordered sets suggest. Instead, different depths in the hierarchy indicate more (or less) involved descriptions or local modeling of the data. Patterns at the same level (having potentially the same predecessor) give users additional information about how they belong together and are influenced by patterns further up in the hierarchy. Decision trees, for instance, are tree sets in which the patterns in each node are single attribute-value pairs. As in the case of ordered sets, redundancy can be controlled more implicitly, making such techniques easier to use. Additionally, since hierarchies usually trade off width against depth, the amount of patterns that has to be considered to interpret one particular pattern is smaller than for ordered sets. Successors are effectively turned into siblings (or even relegated to entirely differently parts of the hierarchy), making the overall picture clearer. The further processing of those sets turns out to be even more restricted than in the case of ordered sets, however.

Algorithmic differences

The final distinction that can be made in pattern set mining is as a difference on how to perform such mining, as we already mentioned above:

1. Mine local patterns first and then perform a post-processing step to select the pattern set from it.

2. Mine local patterns, modify/manipulate the data, repeat the mining iteratively.

The greatest strength of the first approach is that the local pattern mining operation has to be performed just once. Even given the efficiency of state-of-the-art algorithms, such mining operations take time and a post-processing technique will therefore deliver results more quickly. This single local mining run, on the other hand, means that other characteristics are static, such as the background distribution against which patterns are considered interesting, or the language used. Patterns found this way are *interesting with regard to this distribution* and the post-processing operation will be effective if they describe different subsets of the data. Examples include the aforementioned CBA and *pattern teams* formation techniques (Liu et al. 1998, Knobbe and Ho 2006b).

The second approach will perform several local mining operations and, depending on the type of mining performed, those can become rather costly. Due to the manipulation of the data, however, further mining runs find patterns that are *interesting with regard to patterns found before*. Generally, it would also be possible to change the type of pattern language from one level of the hierarchy to the next, for instance increasing the complexity and expressivity. The trade-off between the *post-processing* and *iterative mining* techniques is therefore one between efficiency (in post-processing), and effectiveness and expressivity (in iterative mining). Iterative mining techniques have been described in (Bringmann and Zimmermann 2005, Rückert and Kramer 2007, Geamsakul et al. 2003), for instance.

As this short explanation shows, there is a fundamental difference in the kind of interesting patterns that can be assembled into a set by those two techniques, which is the reason that we decided to structure the thesis along this distinction. Both techniques can be used to induce ordered sets. The modification of data in the second approach, however, means that unordered sets are impossible to induce due to the influence of earlier mined patterns. In a similar vein, although it is theoretically possible to design a post-processing technique that leads to the mining of tree sets, we will see in the discussion that those sets would be of limited meaning. But those are the only limitations. Both approaches can be used in terms of the tasks we have outlined above and secondary considerations, like used pattern languages or constraints that have to be satisfied can also be addressed nicely.

Contributions and Roadmap

The thesis is organized into three main parts, and a fourth part summarizing our findings. The first, and largest, part discusses the theoretical foundations and introduces our framework of constraint-based pattern set mining. It also describes the algorithmic solutions on whose experimental evaluation we report in Parts II and III. Part II evaluates post-processing methods for pattern set mining, comparing existing and novel solutions. Part III evaluates iterative pattern set mining methods, mainly showing how data mining can benefit from this approach. Part IV concludes our work.

Part I Our main contribution in this work is providing a framework of constrained pattern set mining as a distinct data mining task in which sets of patterns are assembled that satisfy user-defined constraints. Such constraints can involve both characteristics of the entire pattern set, such as its size, and of relationships between individual elements of the set (patterns), such as a limit on allowable redundancy among them. In this way, we add to the interestingness that users can specify in terms of validity and novelty by defining constraints on local patterns a method for specifying interestingness in terms of understandability (and usefulness). We also thoroughly discuss the properties if constrained pattern set mining, particularly its relation to local pattern mining. This is to the best of our knowledge the first time that such a definition has been given and we devote Chapter 1 to it.

The basis for pattern set mining is the field of local pattern mining. This is even more true since we aim to make the language and knowledge of local pattern mining applicable to this new field. We therefore define patterns, their properties, and constraints used to allow the user to define which patterns he would find *interesting* with regard to validity and novelty in Section 1.1.

We also shortly outline the KDD tasks we will address in this work, and gives examples in which way patterns and sets of patterns are involved in solving them.

Following that, we introduce our framework in Section 1.3 and work out the similarities to the local pattern mining setting in detail. In this way we can show that much of the knowledge about local pattern mining, such as ways to improve the efficiency of mining techniques can be applied directly to constrained pattern set mining as well. This section is an extension on results published in (De Raedt and Zimmermann 2007).

With that theoretical base established, we can discuss algorithmic solutions in Chapter 2. We show algorithms from both the fields of machine learning and data mining, describe them in terms of the introduced framework, and discuss to what extent they can be used for pattern and pattern set mining. The main contribution of that chapter lies in the introduction of two methods for post-processing sets of local patterns. The first, exhaustive, technique has been published in (De Raedt and Zimmermann 2007). While other exhaustive techniques for pattern set mining exist, they typically include hard-wired size constraints to control computational complexity. Our approach, in contrast, allows the use of arbitrary constraints for pattern set mining. The second, heuristic, technique has been published in (Bringmann and Zimmermann 2009). While similar to existing techniques, we analyze it further show alternatives to existing design choices. Additionally, we systematically distinguish between the local pattern mining and pattern set mining phase, showing that existing approach from machine learning can be used more flexibly than they have been so far.

The question whether such distinction are useful in practice is experimentally answered in Parts II and III. Part I is based on material that has appeared in the following publications:

De Raedt, L. and Zimmermann, A. (2007). *Constraint-based pattern set mining*. In: Proceedings of the Seventh SIAM International Conference on Data Mining.

Bringmann, B. and Zimmermann, A. (2009). *One in a million: Picking the right patterns*. Knowledge and Information Systems, 18(1):6181.

Part II discusses of post-processing techniques for the purpose of pattern set mining. First, we will discuss the mining of unordered sets of patterns in Chapter 3, employing the exhaustive pattern set mining method and the heuristic post-processing method we propose. Since we adapted exhaustive local pattern mining for pattern set mining, we first ask

1. Whether our technique can mine constrained pattern sets?
2. Whether the same algorithmic behavior can be observed?
3. Whether the resulting sets are of better quality than ones mined by existing heuristic methods?

We can answer all three of these questions positively, supporting our discussion. The results in this section have been published in (De Raedt and Zimmermann 2007).

The second method adapts feature selection methods. We employ two different heuristic search strategies, one which has been applied for instance in (Liu et al. 1998, Siebes et al. 2006), and a newly developed one, and experimentally evaluate their effects.

In Chapter 4, we mine ordered set of patterns. All methods used in this chapter are heuristic, depending on orders that are defined on the pattern set to process it. Starting from the hypothesis that orders imposed during local pattern mining are more useful than orders imposed afterwards, we evaluate the composition and quality of derived pattern sets. The results, published in (Zimmermann and Bringmann 2005) and (Zimmermann and De Raedt 2009), validate our hypothesis.

Even orders arising during the local pattern mining step can still be somewhat arbitrary, however, since they mostly rely on user-specified constraints. An order that is imposed by the process, so-to-speak, would seem to have a better chance to lead to compact, high-quality pattern sets. Part II is based on material that has appeared in the following publications:

Zimmermann, A. and Bringmann, B. (2005). *CtC - Correlating tree patterns for classification*. In: Han, J., Wah, B. W., Raghavan, V., Wu, X., and Rastogi, R., editors, Proceedings of the Fifth IEEE International Conference on Data Mining, pages 833836

De Raedt, L. and Zimmermann, A. (2007). *Constraint-based pattern set mining*. In: Proceedings of the Seventh SIAM International Conference on Data Mining.

Bringmann, B. and Zimmermann, A. (2009). *One in a million: Picking the right patterns. Knowledge and Information Systems*, 18(1):6181.

Part III In the third part, we turn our attention towards iterative pattern mining. As we mentioned above, all iterative mining techniques originate in machine learning and have rarely been used with more than one approach to local pattern mining. Our main question in this part is therefore, whether resulting pattern sets can be improved by replacing the heuristic local pattern mining step developed in machine learning by exhaustive techniques developed in data mining.

The first chapter of that part, Chapter 5, considers sequential mining to assemble ordered sets of patterns. We replace the heuristic techniques developed in machine learning with exhaustive techniques and ask whether resulting pattern sets are of better quality. Additionally, we evaluate whether there is a difference in computational complexity between the heuristic and exhaustive approaches. As a final result, we test our hypothesis that iterative mining leads to smaller sets of higher quality when compared to post-processing techniques.

Sequential mining usually assumes that local patterns mined in a single iteration can describe the data satisfactorily. In Chapter 6, we use parallel mining techniques to mine for tree sets of patterns instead. Since these permit the re-description of covered data in more detail by patterns that are mined later, the pressure for highly specific patterns is lessened. We test the hypothesis that parallel mining gives rise to highly compact sets of high quality (work we also published in (Bringmann and Zimmermann 2005)) and find it validated. Our novel contribution to the field in this chapter is the introduction of a new system, CG-Clus, a top-down conceptual clustering algorithm, published in (Zimmermann and De Raedt 2009), and a new formalism, *Ensemble-Trees*, published in (Zimmermann 2008). Employing CG-Clus, we tackle the problem of *conceptual clustering* by a new method, giving a user greater control over the number of formed clusters, and producing a small number of understandable descriptions. *Ensemble-Trees*, finally, point towards the future by introducing a type of pattern set that has not been used so far. Part III is based on work that has appeared in the following publications:

Zimmermann, A. and De Raedt, L. *Cluster-Grouping: From subgoup discovery to clustering*. Machine Learning Journal. *Accepted for publication*.

Bringmann, B. and Zimmermann, A. (2005). $Tree^2$ - *Decision trees for tree structured data*. In: Jorge, A., Torgo, L., Brazdil, P.,

Camacho, R., and Gama, J., editors, PKDD 2005, Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, pages 4658.

Zimmermann, A. (2008). *Ensemble-trees: Leveraging ensemble power inside decision trees.* In Boulicaut, J.-F., Berthold, M. R., and Horvath, T., editors, Discovery Science, 11th International Conference, DS 2008, Proceedings, pages 7687

Part IV is used to summarize the findings of this thesis. We relate the results of our various experiments in a more global manner than the conclusions to individual chapters or parts allowed. Discussing these results, we will draw our final conclusions and point towards future research directions that spring from our work.

Finally, some of the work that was performed during the course of my PhD research has not been included in the text of this thesis and is briefly summarized here. In the context of top-k class-correlated pattern mining, we experimentally evaluated *stepwise* correlated mining in several molecular domains and published our results in (Bringmann et al. 2006). We aimed at exploring the trade-off between expressivity of a pattern language and the quality of mined patterns, by incrementally increasing the complexity of the pattern language. The kth-worst score of the preceding language was used as starting threshold in each local pattern mining operation so that increased complexity had to correspond to higher quality. We showed that stepwise mining is more efficient than mining the top-k patterns in a complex language directly. Additionally, we found that patterns of lower complexity (sequences) can be more suitable as features for concept learning than features of higher complexity (graphs), if the same number of patterns is used. In a recent work (Zimmermann and Bringmann 2009), we explored the use of subsampling techniques for the selection of high-quality features for concept learning. We worked in a molecular domain and evaluated several techniques inspired by machine learning approaches such as bagging, boosting, and decision tree post-pruning. The main result of our experiments was that using non-verlapping subsets of the data and aggregating the resulting result sets proved to be more effective than using the entire data set for mining, which could be exploited in parallelized data mining. We also found that uniformly selecting patterns from the result set lead to good feature sets.

Part I

Foundations

Overview of Part I

In the first part of the thesis, we lay out the fundamentals of our work. We begin by discussing patterns and pattern sets in Chapter 1. Local pattern mining provides the basis of pattern set mining and, as we will show, pattern set mining has many similarities with it. Therefore, we discuss local pattern mining in some detail, touching on pattern languages, the properties of patterns, and the constraints that can be derived to guide local pattern mining.

Local pattern mining and pattern set mining are subtasks of the data mining step in the context of different KDD tasks that we introduce in a general way, giving examples of how mining can be used to address them.

We proceed to introduce our main contribution, a formal framework for pattern set mining, in Section 1.3. To the best of our knowledge, this is the first time that such a definition has been given. We identify the three main types of pattern sets that have been used in data mining and machine learning, namely unordered, ordered and tree sets, and discuss their properties. Additionally, we show how local pattern mining constraints can be lifted to the pattern set level and introduce new constraints for pattern set mining.

This framework allows us to discuss existing algorithms for pattern set mining and identify their properties. As we show in Chapter 2, both exhaustive and heuristic local pattern mining techniques exist, with the former having been developed in data mining and the latter in machine learning. In terms of pattern set mining, the difference is mainly between pattern set mining as post-processing, as practiced in data mining, and iterative pattern set mining, which comprises most approaches from machine learning. Post-processing techniques are mostly heuristic because of the large number of patterns that have to be processed or have explicitly included size constraints to control computational complexity. Describing the issue in terms of our framework, however, makes novel general constraint-based exhaustive post-processing methods theoretically possible.

The iterative techniques developed in machine learning have usually been considered together with particular local pattern mining algorithms. The second major insight derived from our framework is that such systems can be decomposed and the local pattern mining step instantiated by different algorithmic solutions, leading to new systems.

Chapter 1

Patterns, Constraints, and Sets of Patterns

As this work is about mining sets of patterns, this chapter is concerned with laying the foundations for understanding this mining process. Following Fayyad *et al.*, a *pattern* is an interesting phenomenon in the data. Interestingness of individual patterns refers mainly to validity and novelty. While novelty is of course dependent on the background knowledge of a user initiating the data mining process, statistical measures can be used as so-called *objective* measures to identify candidates for novelty.

The means of expressing data are usually influenced by the domain in which they were collected as well as the task for which they are analyzed. In Section 1.1, we therefore define the pattern languages which we will employ and discuss the relation to the underlying data. The usage of a formal language to express patterns allows the systematic enumeration of them, which is necessary for efficient data mining. We discuss how they can be enumerated in Section 1.1.1 and properties of patterns regarding the data in Section 1.1.2. Since the amounts of data mined are usually rather large and mining should happen in an automated way, *constraints* are used to define what are interesting phenomena. Those constraints are based on a posteriori properties, properties of patterns that hold with regard to the data, and that can be quantified by different measures. Constraint properties (Section 1.1.4) are used to guide the mining process and make it more efficiently.

Once patterns have been successfully mined, they form the ingredients for further analysis: as features or rules for classifier construction, for instance, or as descriptions of clusters or subgroups, which are interpreted by human end-users. In Section 1.2, we will therefore give an overview over three typical KDD tasks and discuss in general terms how sets of patterns are employed in addressing them.

Given the typically large number of patterns returned by a mining operation, the selection of *subsets* of the full solution set is needed for the patterns to be

useful and understandable. Accordingly, pattern set mining has always been an important task of KDD approaches, even though it was hardly ever defined as a separate task. The third section of this chapter, Section 1.3, is therefore devoted to the foundations of pattern set mining. To the best of our knowledge, this is the first time that a formal framework for pattern set mining is introduced. This formal framework is important as it allows us to use it for analyzing existing mining techniques, improve them, and devise new ones based on the properties of pattern sets and the measures that can be used to guide the search.

We consider three different types of pattern sets in this work: unordered, ordered, and tree sets. Similarly to pattern languages, these set types allow the efficient enumeration of pattern sets during search. To give the user the opportunity to guide the selection process and obtain interesting *pattern sets*, constraints are used again, this time on the pattern set level. Many of the local pattern constraints as well as their properties transfer to the level of sets but the different nature of pattern sets also allows for new constraints. We discuss therefore the same notions we discussed in the context of local patterns for pattern sets, namely a priori and a posteriori properties, constraints and their characteristics.

1.1 Patterns

To facilitate efficient search for patterns, they are defined in terms of formal languages:

Definition 1.1.1. Formal Language *A* formal language $\mathcal{L}$ *is defined in terms of an alphabet* Σ *and* formation rules *or a* grammar Γ*. The grammars specify how symbols of the alphabet are arranged into* words *of the language. The language is therefore a subset of the power set of the alphabet:* $\mathcal{L} \subseteq 2^{\Sigma}$.

Note that all formal languages include the *empty word* ϵ. In the following, we will define several languages that are used throughout this work in the context of local pattern mining. We start with the language of itemsets:

Definition 1.1.2. Itemsets *Given a set of items* $\Sigma_{\mathcal{I}}$*, the language of itemsets* $\mathcal{L}_{\mathcal{I}}$ *takes the form* $\mathcal{L}_{\mathcal{I}} = 2^{\Sigma_{\mathcal{I}}}$*, the set of all sets of items in* $\Sigma_{\mathcal{I}}$.

More complex in terms of the symbols of the alphabet and of the composition of words is the language of attribute-value conjunctions:

Definition 1.1.3. Attribute-Value Language *Given a set of attributes* $\mathcal{A} = \{A_1, \ldots, A_d\}$*, having values*

$$dom(A_i) = \{v_1^i, \ldots, v_{i_t}^i\},$$

we define the alphabet

$$\Sigma_{AV} = \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq i_t} \{A_i = v_j^i\}.$$

We can now define a family of languages via the application of a rule that describe well-formed words of the language:

$$\mathcal{L}_{AV} = \{p \subset \Sigma_{AV} |\ \textit{each attribute occurs at most once in } p\}$$

A third pattern language that finds widespread application in data mining is that of rooted trees:

Definition 1.1.4. Labeled, Ordered, Rooted Trees *Given an alphabet of* vertex labels Σ_v*, a rooted k-tree t is a set of k nodes* V_t *where each* $v \in V_t$*, except one called* root*, has a parent denoted* $\pi(v) \in V_t$*. We use* $\lambda(v) \in \Sigma_v$ *to denote the label of a node and an operator* $\prec$ *to denote the order from left to right among the children of a node. The transitive closure of* π *will be denoted* π^**. Then* $\mathcal{L}_{tree}$ *denotes the formal language composed of all labeled, ordered, rooted trees .*

In the rest of this thesis, we will use lower-case letters to denote individual patterns, such as p, g, or s. Patterns are derived from *instances*, collected in a *data set* $\mathcal{D} = \{e_1, \ldots, e_n\}$ with each of those instances member of a data language $\mathcal{L}_{\mathcal{D}}$. Generally speaking, every pattern language is also a language for describing data instances. There are exceptions, however: by extending the language introduced in Definition 1.1.3:

$$\mathcal{L}_{\mathcal{A}} = \{p \subset \Sigma_{AV} |\ \text{each attribute occurs exactly once in } p\}, \tag{1.1.1}$$

we introduce a data language that is distinguished by the other three by the fact that each instance has the same number of elements. Itemsets, for instance, can have an arbitrary number of items (limited only by $\Sigma_{\mathcal{I}}$).

To meaningfully speak about the relationship between patterns and instances, it is necessary to define a *matching function*

$$match : \mathcal{L}_p \times \mathcal{L}_{\mathcal{D}} \mapsto \{0, 1\}.$$

Starting again with itemsets, the matching function is the subset relation:

Definition 1.1.5. Itemset Matching *Given a pattern* $p \in \mathcal{L}_{\mathcal{I}}$*, and an instance* $e \in \mathcal{L}_{\mathcal{I}}$

$$match(p, e) = 1 \textit{ if and only if } p \subseteq e$$

In the case of $\mathcal{L}_{AV}$ and $\mathcal{L}_{\mathcal{A}}$, for each attribute in the pattern, this particular attribute's value in an instance is checked:

Definition 1.1.6. Attribute-Value Matching *Given* $p \in \mathcal{L}_{AV}$ *and an instance* $e \in \mathcal{L}_{\mathcal{A}}$*, the matching function is:*

$$match(p, e) = 1 \textit{ if and only if } \forall A_i = v^i_{i_l} \in p : \exists A_i = v^i_{i_l} \in e$$

Finally, for rooted trees, several notions of tree matching exist. We will use the one employed by Zaki *et al.* (2003), *tree embedding*:

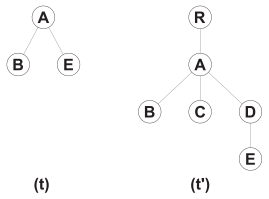


Figure 1.1: The tree t is embedded in t'.

Definition 1.1.7. Tree Embedding *Given a pattern and an instance* $p, e \in \mathcal{L}_{tree}$*, tree* p *is* embedded *in a tree* e *if and only if a mapping* $\varphi : V_p \rightarrow V_e$ *exists such that*

$$\forall u, v \in V_p : \lambda(u) = \lambda(\varphi(u)) \wedge$$
$$u \prec v \Leftrightarrow \varphi(u) \prec \varphi(v) \wedge$$
$$v \in \pi^*(u) \Leftrightarrow \varphi(v) \in \pi^*(\varphi(u)).$$

$$match(p, e) = 1 \text{ if and only if } p \text{ is embedded in } e$$

An example of an embedded tree is given in Figure 1.1. This notion is more flexible than simple subtrees and mining for local tree patterns is still efficient. In general, other matching notions (see (Kilpeläinen 1992)) and even different representations could be used.

As preceding paragraphs show, the techniques we discuss in this work are not limited to particular patterns or data. Therefore, the following sections are applicable for all types of local pattern mining that are known in the literature so far, abstracting from the more specific discussions that are often found in papers in the field.

1.1.1 A priori properties

Having defined patterns and the matching operator, properties of individual patterns and relationships between patterns can be defined. The most fundamental of these relationships is the generality relation:

Definition 1.1.8. Pattern Generality *A pattern* g *is said to be* more general than or equal to *pattern* s*, denoted* $g \preceq s$*, if and only if*

$$\forall e : match(s, e) = 1 \Rightarrow match(g, e) = 1.$$

$$\text{If } g \preceq s \wedge \neg(s \preceq g),$$

then pattern g *is strictly* more general than s*, denoted* $g \prec s$*.*

Since pattern mining/induction is basically search, there is the need to formulate an operator for traversing the pattern space in a principled way. This refinement operator adheres to the grammar of $\mathcal{L}_p$ to ensure that only actual words in $\mathcal{L}_p$ are derived.

Definition 1.1.9. Refinement Operator *A* refinement operator *is a function* $\rho : \mathcal{L}_p \mapsto 2^{\mathcal{L}_p}$. *It is called a* specialization ρ_s (generalization ρ_g) operator *if and only if*

$$\forall p' \in \rho(p) : p \preceq p' (p' \preceq p).$$

A refinement operator can be applied repeatedly, turning it into a piecewise-defined function. A superscript is then used to denote the level n *of the refinement:*

$$\rho^n(p) = \begin{cases} \rho(p) & \text{if } n = 1 \\ \bigcup\limits_{p' \in \rho^{n-1}(p)} \rho(p') & \text{if } n > 1 \end{cases}$$

A specialization (generalization) operator is called complete *if and only if for a most general (most specific) pattern* $p_{mg}(p_{ms})$ *in the language:*

$$\forall p \in \mathcal{L}_p : p \in \rho^\infty(p_{mg}) (p \in \rho^\infty(p_{ms}))$$

A specialization (generalization) operator is called ideal *if and only if*

$$\forall p \in \mathcal{L}_p : \rho(p) = \{s \in \mathcal{L}_p | p \prec s \wedge \neg \exists p' : (p \prec p' \prec s)\}$$
$$(\forall p \in \mathcal{L}_p : \rho(p) = \{g \in \mathcal{L}_p | g \prec p \wedge \neg \exists p' : (g \prec p' \prec p)\})$$

The refinement operator is called optimal*, if and only if for any* $p \in \mathcal{L}_p$ *there exists exactly one sequence* $\epsilon = p_0, p_1, \ldots, p_n = p$ *such that* $h_i \in \rho(h_{i-1})$ *for* $i \in \{1, \ldots, n\}$.

1.1.2 A posteriori properties

Patterns, supposed to be meaningful on the data they are mined from (or at least evaluated on), have characteristics that are not a priori known but only emerge in interaction with the data. Since those properties form the foundation for defining the *interestingness* of patterns, a thorough definition of these characteristics, and the measures that can be based on them, is necessary.

Definition 1.1.10. Coverage and Support *Given a data set D, the coverage of a pattern p is defined as*

$$cov(p, \mathcal{D}) = \{e \in \mathcal{D} \mid match(p, e) = 1\},$$

and its support *as* $sup(p, \mathcal{D}) = |cov(p, \mathcal{D})|$. *We assume that each* $e \in db$ *has a unique identifier* $tid(e)$ *and the set of all identifiers of covered instances* $\{tid(e) \mid e \in cov(p, \mathcal{D})\}$ *is called* tid-set$(p, \mathcal{D})$. *For any two patterns* p, q,

$$sup(p \cup q, \mathcal{D}) = |cov(p, \mathcal{D}) \cap cov(q, \mathcal{D})|.$$

Support is a measure that can be used to assess the *validy* of a pattern since patterns that occur often in the data can be expected to occur in unseen data as well. Two patterns can also be combined via $\Rightarrow$: $p \Rightarrow q$, with the standard meaning of $match(p, e) = 1 \Rightarrow match(q, e) = 1$. This relation, also called a rule, is not necessarily a tautology (distinguishing it from $q \preceq p$) and a common way of expressing the probability that if p matches an instance, q does is well, is *confidence*.

Table 1.1: Contingency table for two patterns p, q

	q	$\neg q$	
p	$x^+ = sup(p \cup q, \mathcal{D})$	$x^- = sup(p \cup \neg q, \mathcal{D})$	$sup(p, \mathcal{D})$
$\neg p$	$sup(\neg p \cup q, \mathcal{D})$	$sup(\neg p \cup \neg q, \mathcal{D})$	$\lvert\mathcal{D}\rvert - sup(p, \mathcal{D})$
	$m = sup(q, \mathcal{D})$	$\lvert\mathcal{D}\rvert - sup(q, \mathcal{D})$	$n = \lvert\mathcal{D}\rvert$

Definition 1.1.11. Confidence *The confidence of a rule $r : p \Rightarrow q$ is defined as*

$$conf(r, \mathcal{D}) = \frac{sup(p \cup q, \mathcal{D})}{sup(p, \mathcal{D})}$$

Confidence is a way of quantifying the co-occurence of two patterns, with high confidence meaning that the occurrence of p implies occurrence of q reliably. However, if $\forall e \in \mathcal{D} : match(q, e) = 1, \forall p : conf(p \Rightarrow q) = 1.0$, giving maximum confidence to a rule that predicts an *expected* phenomenon. Measures that quantify this co-occurence, *correlation measures*, therefore take into account both patterns' presence (and absence) in the entire data set. A pattern is said to *correlate* with a target pattern, if their co-occurrence deviates from an expected value. Such a deviation from expectation is a way of *objectively* assessing the *novelty* of a pattern, in contrast to *subjective* measures that need to account for a user's particular expectations (Silberschatz and Tuzhilin 1996). For a fixed target pattern q, we call a pattern p that correlates with it a *correlated pattern*. For this purpose, occurrence counts of two patterns are organized in a contingency table, such as Table 1.1. We extend the *support* notation in the following way:

$$sup(\neg p, \mathcal{D}) = |\overline{cov(p, \mathcal{D})}| = |\mathcal{D} \setminus cov(p, \mathcal{D})| \text{ with}$$

$$sup(\neg p \cup q) = |\overline{cov(p, \mathcal{D})} \cap cov(q, \mathcal{D})|, \text{ accordingly.}$$

Table 1.1 shows a contingency table using the extended notation. To improve readability, we will use a shorthand notation for support counts, in the following way:

$$x^+ = sup(p \cup q, \mathcal{D}), x^- = sup(p \cup \neg q, \mathcal{D}), m = sup(q, \mathcal{D}), n = |\mathcal{D}|.$$

The superscripts are related to the convention of calling instances in which q is present (absent) positive (negative) instances. This allows us to identify a pattern with its *stamp point*.

Definition 1.1.12. Stamp Point *Given $\mathcal{D}$ and a fixed target pattern q, n and m stay fixed for any pattern p, whose* stamp point *is $sp(p, \mathcal{D}, q) = \langle x^+, x^- \rangle$.*

Given that m and n are fixed for a given data set, the stamp point includes all variables needed to describe the behavior of the pattern in the data. Using the stamp point, we will treat interestingness measures (such as support, confidence or correlation measures such as *information gain*) as functions $\sigma : \mathbb{Z}^2 \mapsto \mathbb{R}$. If the target pattern's identity is clear from the context, we will drop it from the stamp point notation.

Example 1.1.1. *Given $\mathcal{D}, p$, fixed q, $conf(p) = \frac{x^+}{x^+ + x^-}$. For a more elaborate measure, information gain (IG), we first have to define the entropy of a pattern q with regard to $\mathcal{D}$:*

$$ent(q, \mathcal{D}) = -\frac{m}{n} \log \frac{m}{n} - \frac{n-m}{n} \log \frac{n-m}{n}$$

with the complete measure taking the form:

$$\begin{aligned} IG(p, \mathcal{D}) &= ent(q, \mathcal{D}) - \frac{cov(p, \mathcal{D})}{|\mathcal{D}|} ent(q, cov(p, \mathcal{D})) - \frac{\overline{cov(p, \mathcal{D})}}{|\mathcal{D}|} ent(q, \overline{cov(p, \mathcal{D})}) \\ \Leftrightarrow IG(p, \mathcal{D}) &= ent(q, \mathcal{D}) - \frac{x^+ + x^-}{n} \left(-\frac{x^+}{x^+ + x^-} \log \frac{x^+}{x^+ + x^-} - \frac{x^-}{x^+ + x^-} \log \frac{x^-}{x^+ + x^-} \right) \\ &\quad - \frac{n - (x^+ + x^-)}{n} \left(-\frac{m - x^+}{n - (x^+ + x^-)} \log \frac{m - x^+}{n - (x^+ + x^-)} \right. \\ &\quad \left. -\frac{n - m - x^-}{n - (x^+ + x^-)} \log \frac{n - m - x^-}{n - (x^+ + x^-)} \right) \end{aligned}$$

Entropy takes its maximum value when $m = \frac{n}{2}$. Information gain measures how much entropy is reduced by a certain patterns, that is, how "pure" in terms of the presence (absence) of q the subsets are which are covered or not covered by p. A second measure, χ^2, quantifies how often $p, \neg p$ and $q, \neg q$, occur together, respectively. To this end, the *observed* co-occurrence is compared to the *expected* co-occurrence:

Example 1.1.2. *The observed co-occurrences are given by:*

$$O(p, q) = x^+, O(\neg p, q) = m - x^+, O(p, \neg q) = x^-, \text{ and } O(\neg p, \neg q) = n - m - x^-$$

and the expected co-occurrences by:

$$E(p, q) = \frac{(x^+ + x^-) \cdot m}{n}, E(p, \neg q) = \frac{(x^+ + x^-) \cdot (n - m)}{n},$$

$$E(\neg p, q) = \frac{(n - (x^+ + x^-)) \cdot m}{n}, E(\neg p, \neg q) = \frac{(n - (x^+ + x^-)) \cdot (n - m)}{n}$$

Finally, the full measure takes the form:

$$\begin{aligned} \chi^2(p, \mathcal{D}) = \; & \tfrac{(O(p,q) - E(p,q))^2}{E(p,q)} + \tfrac{(O(p,\neg q) - E(p,\neg q))^2}{E(p,\neg q)} + \\ & \tfrac{(O(\neg p,q) - E(\neg p,q))^2}{E(\neg p,q)} + \tfrac{(O(\neg p,\neg q) - E(\neg p,\neg q))^2}{E(\neg p,\neg q)} \end{aligned}$$

We can see that both IG and χ^2 are *symmetrical* correlation measure, that is, x^+ and x^- could be interchanged without leading to a different value. *Negative correlation*, the co-occurrence of p's presence with the absence of q is therefore considered equivalent to *positive correlation*, the co-occurrence of p's presence with q's presence. If patterns are mined for a *discriminative* purpose, such as learning the classification of instances, or assigning instances to certain subsets,

as in clustering, this is effective. If the goal is to connect patterns to a predefined subset, as in *subgroup discovery*, for instance, *positive correlation* is more interesting. A useful measure in this context is *Weighted Relative Accuracy* (*WRAcc*):

Example 1.1.3.

$$WRAcc(p, \mathcal{D}) = \frac{(x^+ + x^-)}{n} \left(\frac{x^+}{x^+ + x^-} - \frac{m}{n} \right)$$

Finally, we can define the concepts of *freeness* and *closedness* for patterns, which we will use extensively during the rest of this work:

Definition 1.1.13. Freeness and Closedness *Given a pattern language $\mathcal{L}_p$, an interestingness measure σ, data set $\mathcal{D}$, a pattern p is called* free *if and only if $\forall g, g \prec p : cov(g) \neq cov(p)$. It is called* closed *iff $\forall s, p \prec s : cov(s) \neq cov(p)$.*

Both pattern types have advantages: *Free* patterns tend to generalize well (match unseen instances) which make them useful for concept learning purposes. Additionally, they are the shortest words in $\mathcal{L}_p$ which are needed to describe certain data; longer words use more elements from Σ_p without increasing the information value. They are therefore useful for descriptive purposes like *subgroup discovery* and *clustering* as well. On the other hand, fewer *closed* than *free* patterns occur in the data, which means that sets of closed patterns are smaller and thus easier understandable.

1.1.3 Constraints

While we have referred to the measures mentioned in the preceding section as *interestingness measures*, interestingness in the end lies in the eye of the beholder, namely the user who is using the pattern mining process. Pattern mining therefore gives the user a way of specifying which patterns he considers interesting, in the form of *constraints*:

Definition 1.1.14. Constraint *A* constraint *c is a predicate on the language of patterns that evaluates to* true *or* false*: $c : \mathcal{L}_p \mapsto \{true, false\}$.*

Constraints are usually based on a priori or a posterior properties of patterns. The set of all patterns that satisfy a constraint c on a given $\mathcal{D}$ is the theory $Th(\mathcal{L}_p, \mathcal{D}, c) = \{p \in \mathcal{L}_p | c(p, \mathcal{D}) = true\}$, see (Mannila and Toivonen 1997). Constraints, being logical atoms, can be subjected to the usual logical operators $\wedge, \vee, \neg$ with the usual semantics.

Example 1.1.4. *The theory in strong association rule mining, given a set of items $\mathcal{I}$ and database $\mathcal{D}$, takes the form $Th(\mathcal{L}_\mathcal{I} \times \mathcal{L}_\mathcal{I}, \mathcal{D}, c) = \{p \Rightarrow q | sup(p \cup q) \geq \theta_s \wedge conf(p \Rightarrow q) \geq \theta_c\}$.*

1.1.4 Properties of constraints

The effective mining of interesting patterns is connected to the use of properties of constraints to *prune* the search space which can be rather large, depending on the size of Σ_p and the grammar used. The most influential of these constraint properties has been (anti-)monotonicity:

Definition 1.1.15. Anti-Monotonicity *A constraint c is called (anti-)monotone with regard to the generality relation $\prec$ if and only if*

$$\forall g, s \in \mathcal{L}_p, g \prec s : c(g) \Rightarrow c(s)(c(s) \Rightarrow c(g)).$$

Example 1.1.5. *The best known anti-monotone constraint is arguably the minimum support constraint*

$$sup(p, \mathcal{D}) \geq \theta_s$$

introduced in (Agrawal et al. 1993). The anti-monotonic property was first used for pruning the pattern space in the Apriori *algorithm* (Agrawal and Srikant 1994)*, jump-starting the field of local pattern mining.*

A weaker form of anti-monotonicity is that of *convertible* (anti-)monotone constraints:

Definition 1.1.16. Convertible Constraint *A constraint c is called* convertible (anti-)monotone *if and only if there exists an order $<_{lex}$ on Σ_p such that for all prefix-ordered patterns $p = i_1 \ldots i_n$ such that $i_1 <_{lex} \ldots <_{lex} i_n$:*

$$c(i_1 \ldots i_n) \Rightarrow c(i_1 \ldots i_{n-1})(c(i_1 \ldots i_{n-1}) \Rightarrow c(i_1 \ldots i_n)).$$

Convertible constraints have so far only been used in the realm of itemset mining, for example, the minimum average price constraint:

Example 1.1.6. *For the identification of each item $i \in \Sigma_{\mathcal{I}}$ with a price: $price : \Sigma_{\mathcal{I}} \mapsto \mathbb{R}^+$, a function denoting the average price of an itemset $avg_{price} : \mathcal{L}_{\mathcal{I}} \mapsto \mathbb{R}^+$ would take the form*

$$avg_{price}(I) = \frac{1}{|I|} \sum_{i \in I} price(i)$$

The minimum average price constraint takes the form $avg_{price}(I) \geq \theta_{price}$ and if items are ordered descending in price, it becomes a convertible anti-monotone constraint.

A third property that has been identified in the literature is that of succinctness. We employ a simpler definition (proposed by Bart Goethals, personal communication) that is equivalent to the original one in (Ng et al. 1998):

Definition 1.1.17. Succinct Constraint *A constraint c is called* succinct *if and only if the constraint $c(p) = true$ can be expressed as $\forall i \in \Sigma, i \in p : r(i) = true$ for some predicate r.*

This type of constraints can be understood as a filter that is applied to patterns from the language:

Example 1.1.7. *If we use the language of rooted trees $\mathcal{L}_{tree}$ for the description of web sessions, we might want to exclude certain parts of a web site from patterns. Let these parts of the site form a set of* labels $\mathbb{L}$. *In that case, a constraint $c(p) : \forall v \in V_p, \lambda(v) \notin \mathbb{L}$ would act as a succinct constraint enforcing this.*

Finally, when the goal of local pattern mining is the mining of *correlated* patterns, it is useful to be able to use the correlation measure itself for pruning:

Definition 1.1.18. Boundable Function *A function $f : \mathcal{L}_p \mapsto \mathbb{R}$ is said to be* upper-boundable (resp. lower-boundable) *with regard to a pattern language $\mathcal{L}_p$ if given $f(p) = k$, $k' \in \mathbb{R}$ can be derived s.t. $f(p') \leq k'$ (resp. $f(p') \geq k'$) for either all $p' \prec p$ or $p \prec p'$.*

Note that we do *not* call a function boundable if k' corresponds to a global maximum or minimum for all p'. If a function is upper-boundable for $p' \prec p$ (resp. $p \prec p'$), it is said to be *generalization* (resp. *specialization*) upper-boundable, and the dual property holds for lower-boundable. It should be noted that given the nature of matching functions in local pattern mining, functions are usually not generalization boundable. We will discuss the necessary conditions for a function to be boundable, and the methods for calculating this upper bound in Section 2.1.1.

1.2 KDD Tasks

Pattern mining and pattern set mining have to be seen in the context of the data mining step of KDD. Generally, pattern mining is concerned with mining the local patterns that are used in a variety of ways, depending on the KDD task at hand. Pattern set mining, on the other hand, selects subsets of local patterns, or directs the local pattern mining search.

The goal of the KDD process can be either *predictive* and *descriptive*. The classical predictive task is that of *concept learning* which we characterize in Section 1.2.1. Descriptive mining can take different forms and in this work we consider *subgroup discovery* (Section 1.2.2), and *clustering* (Section 1.2.3). The divide between these two fields is not as clear-cut as we make it appear here, however, since for instance a clustering can be used to predict attribute values of instances, depending on the cluster they have been assigned to.

1.2.1 Concept learning

Concept learning (or classification) is concerned with instances that are annotated with a (class) label from a set of labels $C = \{c_1, \ldots, c_c\}$: $\langle e_i, c_i \rangle$. The assumption in concept learning is that there is an underlying function that assigns a class-label to a member of the data language: $f : \mathcal{L}_{\mathcal{D}} \mapsto C$. The goal is

then to use labeled data to approximate this function and use this approximation for the prediction of labels of unseen instances.

Patterns can either be used directly for classification, turning the class label into target patterns of the form $C = c_i$. Multi-class problems can be transformed into binary problems:

- One-against-one: classifiers are learned for each *pair* of class labels, with the final classification an aggregation of individual classifers
- One-against-all: classifiers are learned for *each* class label, contrasted with the set of *all* others

In this way the formulation of binary target patterns that we have given before (see Section 1.1.2) can be used to describe concept learning. In concept learning, the term confidence is rebranded training accuracy since it reflects the confidence of the rule on the *training data*.

Example 1.2.1. *Consider for instance a rule learning system like* CN2*: in the local pattern mining step, a predictive rule is mined, and the data is then manipulated by* sequential covering *to lay the foundations for further rules to be added to the set. The goal is to limit redundancy among rules and ensure that rules stay highly predictive. We will discuss the sequential covering mechanism in detail in Section 2.3.1, and evaluate pattern sets mined in this way in Section 5.1.*

A second option is using patterns as features to describe data in a different way from the original data language. It has the advantage that more powerful learning algorithms can be used to approximate the concept function. On the other hand, such a classifier is often harder to interpret.

Example 1.2.2. *A common approach in predicting effects of drugs in silico depends on mining structured patterns from molecules, for instance sequences or graphs. The molecules can then be represented in a way that makes them processable by machine learning techniques. It is desirable to reduce the number of patterns (and therefore the number of features) to decrease the running time of classifier learning and evaluation, and avoid over-fitting effects. We discuss an algorithm for mining a set of good features in Section 3.2*

1.2.2 Subgroup discovery

Similarly to concept learning, *subgroup discovery* is concerned with instances that are assigned labels, which denote membership in a certain group. In contrast to concept learning, however, the goal is not to approximate the concept function but instead to find a *description* of different subgroups of interest.

Example 1.2.3. *A usual example would be a data set $\mathcal{D}$ in which instances are described in terms of physical symptoms and subgroups are labeled with certain diseases. Subgroup discovery will be used to find the symptoms characterizing a disease, to help with recognizing it, for instance.*

While in concept learning the goal is often to mine patterns that are highly accurate, *subgroup discovery* is more concerned with finding subgroup descriptions that have *high generality*, that is, a large cover, and *distributional unusualness* with regard to the target attribute. This difference is reflected in the choice of interestingness measures which give more weight to generality of patterns than in concept learning approaches.

Example 1.2.4. *It is possible that a subgroup can be described in different ways, or that there is a description for a subset of a subgroup which has already been discovered. To prevent that too many redundant subgroup descriptions are returned, it is necessary to direct the search into unexplored areas of the data. We will evaluate ways of doing this in Section 5.2.*

1.2.3 Clustering

Clustering, finally, is concerned with inducing a *partition* on $\mathcal{D}$:

Definition 1.2.1. Partition *A* partition *of $\mathcal{D}$ is a set $P = \{\mathcal{D}_1, \ldots, \mathcal{D}_p\}$ of subsets of $\mathcal{D}$: $\forall \mathcal{D}_i \in P : \mathcal{D}_i \subset \mathcal{D}$. Each element of P is a* cell *of the partition. Furthermore, the elements of P are pairwise disjoint: $\forall \mathcal{D}_i, \mathcal{D}_j \in P, \mathcal{D}_i \neq \mathcal{D}_j : \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$, and their union is $\mathcal{D}$: $\bigcup_{\mathcal{D}_i \in P} \mathcal{D}_i = \mathcal{D}$.*

Members of each cell of P are supposed to be *similar* according to some user-defined criterion, and cells are called *clusters* in the context of clustering. For inducing a partition, the concept of an equivalence relation is useful:

Definition 1.2.2. Equivalence Relation *An* equivalence relation $\sim$ *is a binary relation between two elements of a set which is reflexive, symmetric and transitive:*

$$\forall a \in A : a \sim a$$
$$\forall a, b \in A : a \sim b \Rightarrow b \sim a$$
$$\forall a, b, c \in A : a \sim b \wedge b \sim c \Rightarrow a \sim c$$

An existing partition induces an equivalence relation:

$$\sim_P \equiv \forall e_i, e_j \in \mathcal{D} : e_i \sim_P e_j \Leftrightarrow (\exists \mathcal{D}_k \in P : e_i \in \mathcal{D}_k \wedge e_j \in \mathcal{D}_k).$$

An equivalence relation gives rise to the concepts of the *equivalence class* and the *quotient set*:

Definition 1.2.3. Equivalence Class *Given an equivalence relation $\sim$, the set $[a] =: \{b \in A : b \sim a\}$ is called an* equivalence class*. The set of all possible equivalence classes of A by $\sim$, denoted $A/\sim =: \{[a] : a \in A\}$ is called the* quotient set *of A by $\sim$.*

This means that an equivalence relation $\sim$ *partitions* the data into the equivalence classes, with $P = \mathcal{D}/\sim$.

The descriptive part of clustering lies in analyzing the induced clusters. After partitioning data, each cluster can be analyzed as to can what is *typical* for instances assigned to this cluster. Especially in the case of *conceptual clustering*, the desired description is expressed in terms of items, nominal features, or structures in the data.

1.3 Pattern Sets

Local patterns describe local phenomena and in doing so allow to understand subsets of the data. The final goal of KDD is the description or modeling of the entire data set, to solve the tasks outlined in Section 1.2. Achieving this goal is made more difficult by the fact that local pattern mining algorithms easily return thousands of patterns, maybe even millions. Many of these patterns are redundant, describing the same subspaces of the data in different ways. This makes it impossible for human end users to identify relevant patterns and automated (machine learning) techniques that use the patterns further pay with increased computational effort or reduced quality of solutions.

As we have stated before, the traditional *local pattern mining* task that is addressed in data mining is that of finding a theory

$$Th(\mathcal{L}_p, \mathcal{D}, c) = \{p \in \mathcal{L}_p \mid c(p, \mathcal{D}) \text{ is true}\},$$

where $\mathcal{D}$ is a database, $\mathcal{L}_p$ a language of patterns, and $c(p, \mathcal{D})$ a selection predicate that states the constraints under which the pattern p is a solution w.r.t. the database $\mathcal{D}$.

A large amount of primitives and constraints have been developed to be used on a variety of pattern languages, often used in dedicated algorithms optimized for those particular constraints and languages. The expressiveness of such approaches can be further increased by combining primitives logically, such as in the query Q_0:

$$\begin{aligned} sup(p, Actives) > 0.05 \quad &\wedge \\ sup(p, Inactives) < 0.01 \quad &\wedge \\ C - H - N \preceq p \quad & \end{aligned} \tag{1.3.1}$$

which combines anti-monotonic and monotonic constraints. This query searches in a database $\mathcal{D}$ of active and inactive molecules for patterns that contain a $C - H - N$ group (more formally: that are more specific than the pattern $C - H - N$), are frequent on the *Actives* and infrequent on the $Inactives$.

The approach taken in this section is to integrate this step of data mining, where local patterns are being queried, with an *explicit* additional step, in which *pattern sets* are being queried. Given a pattern language $\mathcal{L}_p$ and a data set, pattern set mining consists of finding the theory of interesting *sets of patterns*:

$$M = Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C}) = \{\mathbb{S} \subseteq 2^{\mathcal{L}_p} \mid \mathcal{C}(\mathbb{S}, \mathcal{D}) = true\}.$$

The resulting set of pattern sets M is obtained by formulating a constraint $\mathcal{C}$ that has to hold for pattern sets, that is, subsets of $2^{\mathcal{L}_p}$, of interest w.r.t. the database $\mathcal{D}$.

1.3.1 Types of pattern sets

Similarly to the pattern languages described in Section 1.1, there are different kinds of pattern sets. The most flexible type of pattern set is the *unordered* set:

Definition 1.3.1. Unordered Set *An* unordered *pattern set is a set of patterns* $\mathbb{S} = \{p_1, \ldots, p_k\} \subseteq \mathcal{L}_p$*, using the usual set definition.*

The unordered nature of those sets means that they can be enumerated effectively and also that there is no minimal, maximal or *ith* element in the set. All of these characteristics exist in *ordered* pattern sets however:

Definition 1.3.2. Ordered Set *An* ordered *pattern set* $(\mathbb{S}, \trianglelefteq)$ *is a set of patterns* $\mathbb{S} = \{p_1, \ldots, p_k\} \subseteq \mathcal{L}_p$ *on whose elements a total binary relation* $\trianglelefteq$ *is defined, that is, this relation satisfies antisymmety, transitivity and totality:*

- $\forall p_1, p_2 \in \mathbb{S} : (p_1 \trianglelefteq p_2 \wedge p_2 \trianglelefteq p_1) \Rightarrow p_1 = p_2$
- $\forall p_1, p_2, p_3 \in \mathbb{S} : (p_1 \trianglelefteq p_2 \wedge p_2 \trianglelefteq p_3) \Rightarrow p_1 \trianglelefteq p_3$
- $\forall p_1, p_2 \in \mathbb{S} : p_1 \trianglelefteq p_2 \vee p_2 \trianglelefteq p_1$

Reflexivity is implied by the other properties. This relation is called a total order*.*

An example for such an order that we will encounter later on, is the order used by the CBA algorithm:

Example 1.3.1. *Given two patterns* $p \Rightarrow q, p' \Rightarrow q'$*, we say that* $p \Rightarrow q \trianglelefteq_{CBA} p' \Rightarrow q'$ *if and only if:*

- $conf(p \Rightarrow q) > conf(p' \Rightarrow q')$ *or*
- $conf(p \Rightarrow q) = conf(p' \Rightarrow q') \wedge sup(p \Rightarrow q) > sup(p' \Rightarrow q')$ *or*
- $conf(p \Rightarrow q) = conf(p' \Rightarrow q') \wedge sup(p \Rightarrow q) = sup(p' \Rightarrow q') \wedge |p| < |q|$ *or*
- $conf(p \Rightarrow q) = conf(p' \Rightarrow q') \wedge sup(p \Rightarrow q) = sup(p' \Rightarrow q') \wedge |p| = |q|$ *and* p *occurs lexicographically before* p'

This example is rather typical in that the totality of the order is achieved by arbitrarily breaking ties (often based on the lexicographical order of Σ_p). *Tree* sets remove the condition of totality on the entire set (and the arbitrariness that is employed for enforcing it):

Definition 1.3.3. Tree Set *A* tree *pattern set* $(\mathbb{S}, \trianglelefteq)$ *is a set of patterns* $\mathbb{S} = \{p_1, \ldots, p_k\} \subseteq \mathcal{L}_p$ *on whose elements a* partial order $\trianglelefteq$ *is defined, that is, a binary relation that is reflexive, antisymmetric and transitive:*

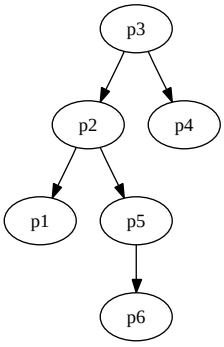


Figure 1.2: A labeled tree representing the tree set from Example 1.3.2

- $\forall p \in \mathbb{S} : p \trianglelefteq p$
- $\forall p_1, p_2 \in \mathbb{S} : (p_1 \trianglelefteq p_2 \wedge p_2 \trianglelefteq p_1) \Rightarrow p_1 = p_2$
- $\forall p_1, p_2, p_3 \in \mathbb{S} : (p_1 \trianglelefteq p_2 \wedge p_2 \trianglelefteq p_3) \Rightarrow p_1 \trianglelefteq p_3$

such that for each $p \in \mathbb{S}$ *the set* $\{p_i \mid p_i \trianglelefteq p\}$ *is totally ordered. Two patterns* p_1, p_2 *are said to be at the same level of the tree set if and only if*

$$|\{p_i \mid p_i \trianglelefteq p_1\}| = |\{p_i \mid p_i \trianglelefteq p_2\}|.$$

As can be seen from the definitions, a total order is a partial order with the added condition of totality (since *totality* implies *reflexivity*). This means that ordered sets can be represented by tree sets but not vice versa.

The formal definition given above aims primarily at generality and to make the relation to ordered sets clear. There is also a less formal way of considering tree sets, namely, as the name implies, in relation to the language of rooted labeled trees we introduced in Definition 1.1.4: *A* tree *pattern set* $(\mathbb{S}, \trianglelefteq)$ *is a labeled rooted tree in which the root is labeled with* $p_r = \inf \mathbb{S}$. *Additionally,* $\forall p_i, p_j \in \mathbb{S} : p_i \trianglelefteq p_j$ *if and only if* $p_j \in \pi^*(p_i)$. Tree pattern sets have the property that $\forall p_i \in \mathbb{S} : |\{p|\pi(p) = p_i\}| \leq 1$.

Example 1.3.2. *Let us assume a tree set* $\mathbb{S} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ *with the patterns ordered according to*

$$p_3 \trianglelefteq p_2, p_3 \trianglelefteq p_4, p_2 \trianglelefteq p_1, p_2 \trianglelefteq p_5, p_5 \trianglelefteq p_6$$

Then $\inf \mathbb{S} = p_3$ *and* $p_3 = \pi(p_2), p_3 = \pi(p_4), p_2 = \pi(p_1), p_2 = \pi(p_5), p_5 = \pi(p_6)$, *leading to the tree shown in Figure 1.2.*

1.3.2 A priori properties

As is the case for local patterns, pattern sets have certain a priori properties. It is often useful to interpret a pattern set $\mathbb{S} = \{p_1, \cdots, p_n\}$ as the disjunction

$p_1 \vee \cdots \vee p_n$ of the patterns p_i it contains. This is reasonable as pattern domains that are being employed can often be considered conjunctions, which means that the *entire* pattern has to appear in an instance for a match to occur. This is consistent with the three matching notions we defined in Section 1.1. From this perspective, a pattern set $\mathbb{S}$ is a kind of formula in disjunctive normal form:

Definition 1.3.4. Pattern Set Matching *A pattern set $\mathbb{S}$ matches an instance e only if there exists a pattern $p \in \mathbb{S}$ that matches e; that is, $match(\mathbb{S}, e) = 1$ only if $\exists p \in \mathbb{S} : match(p, e) = 1$.*

Example 1.3.3. *If the pattern language were reduced to that of individual items, an unordered pattern set would in appearance equal an itemset. The difference between the two is then purely semantical: An itemset is a* conjunction *of items that define a local boundary between two subsets of the data. A pattern set of items is a* disjunction *of items, that collects a number of local boundaries (defined by the individual items) into a larger-scale, boundary.*

Similarly, there is a relation between ordered sets and sequences, or tree sets and tree-structured patterns.

For local patterns, matching has to be defined for any combination of pattern and data language. Pattern set matching, on the other hand, refers to the local pattern definition and therefore holds in this formulation, no matter which languages are used. Note, however, that the definition only gives the necessary condition for a pattern set to match, that there must be at least one pattern in the set that matches the instance. The sufficient condition for a pattern set to match depends on the type of pattern set.

For unordered sets, the condition is intuitively appealing. Patterns are checked against instances in arbitrary order and if at least one matches, the entire set matches. The necessary condition is also sufficient. A side-effect of this is that it cannot be predicted which of several matching patterns will be the one deciding on the match of the entire set.

When looking at ordered sets, such as the decision lists assembled by CBA, it becomes clear that this notion holds there as well. The order is not arbitrary, however, but patterns are checked in ascending order, with the minimal pattern among those matching the instance triggering the matching of the entire set.

Finally, tree sets are generated by parallel mining, described in Section 2.3.3. We will not describe the full algorithm here but state that whenever a pattern p is added to the set, the data is split into subsets $cov(p, \mathcal{D})$ and $\overline{cov(p, \mathcal{D})}$. Each of the two subsets then informs the selection of the next patterns. Hence, of the two direct successors $p_1, p_2 : p = \pi(p_i)$, p_1 will, for instance, match *some* instances in $cov(p, \mathcal{D})$, while p_2 will match *some* instances in $\overline{cov(p, \mathcal{D})}$. Therefore $\forall e \in \mathcal{D} : match(p_1, e) \Rightarrow match(p, e)$. If a p is checked against an instance and matches, the entire pattern set matches. If it does not, it is unintuitive to check whether p_1 matches, and only p_2 and its successors are checked against the instance, until either a match is found or the set is exhausted.

By adopting this notion of matching, the definitions of *cov*, *tid* and *sup* directly carry over to the level of pattern sets. While other notions that apply

to local patterns can be *easily* lifted towards pattern sets, this is not always *trivial*. One of those is the notion of generality which deserves some more attention.

We could directly apply the definition of generality and write that $\mathbb{S} \preceq \mathbb{S}'$ if and only if for all possible instances e it holds that $match(\mathbb{S}', e) \Rightarrow match(\mathbb{S}, e)$. We will refer to this notion of matching as *semantic* generality. The problem with this is, however, that deciding $\mathbb{S} \preceq \mathbb{S}'$ may be non-trivial. For instance, in case of $\mathbb{S}$ and $\mathbb{S}'$ being logical formulae (e.g., in disjunctive normal form as sketched above), this would amount to deciding whether $\mathbb{S}'$ logically entails $\mathbb{S}$, i.e., whether $\mathbb{S}' \models \mathbb{S}$.

At the same time, there are settings in which the set of patterns from which one can assemble pattern sets may be limited, for instance when $\mathbf{L} = Th(\mathcal{L}_p, \mathcal{D}, c)$ and pattern set mining is done in a post-processing step. Then it will be much harder to define refinement operators that compute minimal generalizations or specializations of pattern sets. Therefore, we will adopt an easier but more operational definition of generality at the pattern set level. More formally,

Definition 1.3.5. Syntactic Generality *Pattern set $\mathbb{G}$ is (syntactically) more general than pattern set $\mathbb{S}$, notation $\mathbb{G} \preceq_s \mathbb{S}$, if and only if for all patterns $q_i \in \mathbb{S}$ there exists a pattern $p_j \in \mathbb{G}$ such that $p_j \preceq q_i$.*

It is easy to see that this definition is sound, but – depending on the pattern domain – not necessarily complete, i.e., whenever $\mathbb{S} \preceq_s \mathbb{S}'$ this implies that for all instances e, $match(\mathbb{S}', e) \Rightarrow match(\mathbb{S}, e)$, but the reverse direction does not necessarily hold.

When manipulating pattern sets, it will sometimes be convenient to further simplify this notion of generality and use the simple subset relation.

Definition 1.3.6. Subset Generality *Using the subset relation, $\mathbb{S}$ is (subset) more general than $\mathbb{S}'$, notation $\mathbb{S} \preceq_\subseteq \mathbb{S}'$ if and only if $\mathbb{S}' \subseteq \mathbb{S}$.*

It is instructive to observe that $\mathbb{S} \preceq_\subseteq \mathbb{S}'$ implies that $\mathbb{S} \preceq_s \mathbb{S}'$, i.e., that $\mathbb{S}$ is more general than $\mathbb{S}'$, although the opposite does not always hold. Consider for instance, in the item set domain: for pattern sets $\{\{a\}\} \preceq \{\{a, b\}\}$ holds but $\{\{a\}\} \preceq_\subseteq \{\{a, b\}\}$ does not.

We can define refinement operators accordingly:

Definition 1.3.7. Pattern Set Refinement Operator *A refinement operator P at the pattern set level is a function $P : 2^{\mathcal{L}_p} \mapsto 2^{2^{\mathcal{L}_p}}$. All notations as introduced in Definition 1.1.9 apply for the pattern set level as well.*

Using Definition 1.3.5, we can make the refinement operator explicit:

Definition 1.3.8. Syntactic Refinement Operator *A generalization (specialization) operator in the sense of Definition 1.3.5 takes the form:*

$$P_g(\mathbb{S}) = \{\mathbb{S} \cup \{p'\} \setminus \{p\} \mid p' \in \rho_g(p), p \in \mathbb{S}\} (P_s(\mathbb{S}) = \{\mathbb{S} \cup \{p'\} \setminus \{p\} \mid p' \in \rho_s(p), p \in \mathbb{S}\})$$

Analogously, we can define refinement operators related to Definition 1.3.6:

Definition 1.3.9. Subset Refinement Operator *A generalization (specialization) operator in the sense of Definition 1.3.6 takes the form:*

$$P_{\subseteq_g}(\mathbb{S}) = \{\mathbb{S} \cup \{p\} \mid p \in 2^{\mathcal{L}_p}\}(P_{\subseteq_s}(\mathbb{S}) = \{\mathbb{S} \setminus \{p\} \mid p \in \mathbb{S}\})$$

There is an interesting property that can be exploited to characterize the relationship between $\preceq_s$ and $\preceq_\subseteq$, based on the notion of a reduced pattern set.

Definition 1.3.10. Reduced Pattern Sets *A pattern set $\mathbb{S}$ is reduced if and only if $\forall p \in \mathbb{S} : \neg[(\mathbb{S} \setminus \{p\}) \preceq_s \mathbb{S}]$. We use the notation reduced($\mathbb{S}$) to signify that $\mathbb{S}$ has this property.*

Reduced pattern sets contain no redundant patterns, that is, no pattern that does not affect the level of generality. Given a non-reduced pattern set $\mathbb{S}$, it is easy to obtain a pattern set $\mathbb{S}'$ that is equivalent (w.r.t. $\preceq_s$) by repeatedly deleting patterns p from $\mathbb{S}$ for which $(\mathbb{S} \setminus \{p\}) \preceq_s \mathbb{S}$. The reduced set will be unique provided that the pattern language does not contain syntactic variants.

At this point, the reader commonly working with item sets may notice that the direction of generality is reversed here. Indeed, when working at the local pattern level of item sets, $p \subseteq q$ implies that $p \preceq q$. To see why this is the case, it is – again – convenient to recall the view of pattern sets as formulae in disjunctive normal form. No matter whether one is working with individual patterns or pattern sets, in this view, $\mathbb{G} \preceq \mathbb{S}$ if and only if $\mathbb{S} \models \mathbb{G}$. Subset generality will be convenient to generate pattern sets in mining operations since, when using the subset relation, generalization corresponds to adding a pattern, and specialization to deleting one.

1.3.3 A posteriori properties

Contrary to constraints defined on individual patterns, which determine the *interestingness* of those, pattern sets are evaluating patterns in terms of the relationships with other patterns in the pattern set. Measures and constraints on pattern sets thus focus on pairs of patterns or (implicitly or explicitly) on all patterns collected in the pattern set.

Definition 1.3.11. Pairwise and Set Measures *A measure $\Phi : \mathcal{L}_p \times \mathcal{L}_p \mapsto \mathbb{R}$ is called a* pairwise measure *and a measure $\Psi : 2^{\mathcal{L}_p} \mapsto \mathbb{R}$ is called a* pattern set measure.

In this section, we introduce some useful primitives for pattern set mining and discuss their properties. The list is by no means exhaustive; it merely serves to illustrate the framework and its intuitive appeal.

Definition 1.3.12. Redundancy *Given two patterns p_1, p_2, their* overlap *in a data set $\mathcal{D}$ is defined as*

$$ovlp(p_1, p_2, \mathcal{D}) = cov(p_1, \mathcal{D}) \cap cov(p_2, \mathcal{D}).$$

Their redundancy *in* $\mathcal{D}$ *is defined as*

$$red(p_1, p_2, \mathcal{D}) = |ovlp(p_1, p_2, \mathcal{D})|.$$

Their relative *redundancy is*

$$red_{rel}(p_1, p_2, \mathcal{D}) = \frac{red(p_1, p_2, \mathcal{D})}{|\mathcal{D}|}.$$

The redundancy is a measure of the degree to which two patterns overlap. Like support, it can be defined either absolutely or relatively to the size of the data set $\mathcal{D}$.

This becomes important in cases in which e.g. a characterization of a dataset is wanted, consisting of patterns showing little redundancy which thus highlight characteristic properties of certain subsets. Similarly, when using an unordered rule set, little redundancy means few instances which might be classified differently by different rules, reducing the need for conflict solution heuristics.

Similarly to overlap, symmetric difference is a useful primitive relating two patterns (kindly suggested by Siegfried Nijssen, personal communication):

Definition 1.3.13. Distinctiveness *Given two patterns* p_1, p_2*, the set*

$$\text{diff}(p_1, p_2, \mathcal{D}) = \{cov(p_1, \mathcal{D}) \cup cov(p_2, \mathcal{D})\} \setminus ovlp(p_1, p_2, \mathcal{D})$$

is called their symmetric difference. *Their* distinctiveness *is*

$$dist(p_2, p_2, \mathcal{D}) = |\text{diff}(p_1, p_2, \mathcal{D})|.$$

Dual to redundancy, distinctiveness quantifies how much of the covered data is *exclusively* covered by one of the two patterns. Again, one could define the relative distinctness $dist_{rel}$, as a variant.

Especially for settings such as subgroup discovery, the existing data is often split into subsets corresponding to interesting groups. For pattern sets mined in such a setting, predicates that are defined with regard to the different subsets have to be used for effective mining. This will very likely lead to a situation in which boolean combinations of predicates have to be used for mining and pruning, probably with differing properties. In this context, measures such as the representativeness are useful.

Definition 1.3.14. Representativeness *Given a pattern set* $\mathbb{S}$ *and database* $\mathcal{D} = \{D_1, ..., D_n\}$*, the* representativeness *of* $\mathbb{S}$ *with regard to a given* D_k *is defined as*

$$rep(\mathbb{S}, D_k, \mathcal{D}) = \frac{sup(\mathbb{S}, D_k)}{sup(\mathbb{S}, \mathcal{D})}.$$

The representativeness of a pattern set indicates how characteristic the examples covered by the pattern set are for the subset D_k. If membership in the data set D_k would be represented as an item or attribute, the representativeness would correspond to the confidence of a DNF rule with this item as right-hand side.

1.3.4 Constraints

Analogously to the local pattern mining approach, constraints can be defined that pattern sets of interest have to satisfy.

Definition 1.3.15. Pairwise and Pattern Set Constraint *A predicate* $C : \mathcal{L}_p \times \mathcal{L}_p \mapsto \{true, false\}$ *is called a* pairwise constraint *on patterns and a predicate* $\mathcal{C} : 2^{\mathcal{L}_p} \mapsto \{true, false\}$ *a pattern set constraint.*

The first type of constraints are direct adaptations of the well-known constraints for local pattern mining. In particular, some standard constraints include

Minimum support	$sup(\mathbb{S}, \mathcal{D}) > \theta_s$	(1.3.2)
Maximum support	$sup(\mathbb{S}, \mathcal{D}) < \theta_s$	(1.3.3)
Minimum size	$size(\mathbb{S}) > \theta_{card}$, where $size(\mathbb{S}) = \|p\|$	(1.3.4)
Maximum size	$size(\mathbb{S}) < \theta_{card}$, where $size(\mathbb{S}) = \|p\|$	(1.3.5)
More general than	$\mathbb{S}' \preceq \mathbb{S}$, where $\mathbb{S}$ is a particular pattern set	(1.3.6)
More specific than	$\mathbb{G} \preceq \mathbb{S}'$, where $\mathbb{G}$ is a particular pattern set	(1.3.7)

Size, redundancy, distinctiveness and representativeness are primitives of the constraint language. Typically, these primitives will not be employed in an isolated fashion, but rather they will be combined with aggregates. These aggregates range over the whole set of patterns in the set or, in the case of pairwise primitives, over the set of pairs of patterns in the set. We shall be using the typical aggregates such as avg, min, sum and max. For instance, the constraint

$$max(sup(\mathbb{S}, D)) < \theta_s$$

denotes that the maximum support of any pattern in $\mathbb{S}$ should be less than θ_s. So, it is actually an abbreviation for

$$\max_{p \in \mathbb{S}}(sup(p, \mathcal{D})) < \theta_s$$

Similarly, the constraint

$$sum(red(\mathbb{S}, \mathcal{D})) < \theta_{red}$$

denotes that the sum, taken over all pairs of patterns in $\mathbb{S}$, of the redundancies should be less than θ_{red}. More formally, this amounts to

$$\sum_{i<j, p_i, p_j \in \mathbb{S}} red(p_i, p_j, \mathcal{D}) < \theta_{red}$$

Similarly but less selective are the use of the universal

$$all(red(\mathbb{S}, \mathcal{D})) \leq \theta_{red} \Leftrightarrow \forall p_i, p_j \in \mathbb{S} : red(p_i, p_j, \mathcal{D}) \leq \theta_{red}$$

and the existential quantifier

$$exists(red(\mathbb{S}, \mathcal{D})) \leq \theta_{red} \Leftrightarrow \exists p_i, p_j \in \mathbb{S} : red(p_i, p_j, \mathcal{D}) \leq \theta_{red}$$

A special position is reserved for top-k constraints, since this particular constraint on the one hand has a different mechanism – involving the entire pattern set as well relationships between patterns, but not in an explicit way – and on the other hand is almost indispensable for most model building.

Definition 1.3.16. Top-k Theory *Given an interestingness measure σ, data set $\mathcal{D}$, and pattern language $\mathcal{L}_p$, a top-k theory is a set $Th(\mathcal{L}_p, \mathcal{D}, \sigma) \subseteq \mathcal{L}_p$, $|Th(\mathcal{L}_p, \mathcal{D}, \sigma)| = k$ such that:*

$$\textit{there is no } q \in \mathcal{L}_p : q \notin Th(\mathcal{L}_p, \mathcal{D}, \sigma) \textit{ and } \sigma(q, \mathcal{D}) > \min_{p \in Th(\mathcal{L}_p, \mathcal{D}, \sigma)} \sigma(p, \mathcal{D})$$

We denote the corresponding constraint with $\arg_k \max \sigma(p, \mathcal{D})$.

A top-k constraint obviously evaluates any pattern's satisfactoriness with regard to the data and *all* other patterns in the language. So while it is obviously a pattern set constraint, it is not a pairwise constraint and cannot be defined in terms of pairwise constraints.

To illustrate the way our framework can be employed, we present some example queries here that are meant to illustrate the expressiveness and the use of constraint-based pattern set mining. All the queries sketched here assume that a previous query at the *local pattern mining* level has already been formulated. As one possible such query, it may be convenient to keep the query Q_0 in mind that was introduced in Query 1.3.1. The pattern set variable $\mathbb{S}$ then ranges over all subsets of $Th(\mathcal{L}_p, Mol, Q_0)$, where the database Mol of molecules is composed of two disjoint sets: Act, the actives, and $InAct$, the inactives.

First, in a summarization or clustering context, the user might pose Q_1:

$$sup(\mathbb{S}, Act) \geq \theta_s \wedge max(red(\mathbb{S}, Mol)) \leq 1 \tag{1.3.8}$$

It asks for those sets of patterns that together cover at least θ_1 active molecules and in which the conjunction of each pair of patterns covers at most *one* example. The minimum support ensures that the pattern collection is really representative of the dataset while the maximum redundancy, as mentioned above, means that individual patterns capture characteristics particular to certain distinct subsets.

Second, when focusing on classification or accuracy, one might pose Q_2:

$$\begin{aligned} all(rep(\mathbb{S}, Act, Mol)) \geq 0.95 \wedge \\ max(red(\mathbb{S}, Mol)) \leq \theta_{red} \wedge \\ size(\mathbb{S}) \geq 2 \end{aligned} \tag{1.3.9}$$

Q_2 generates sets of patterns of size at least two, in which the representativeness, (i.e., the confidence in predicting an active) is at least 95%, and the number of examples covered by the intersection of any pair of patterns is at

most θ_{red}. This type of query is related to the process of subgroup discovery, since a particular target class is specified.

Third, our chemical expert suggested query Q_3, which models a form of chemical interestingness:

$$\begin{aligned} \mathbb{S} \preceq \{p_1\} \wedge \quad & sup(\mathbb{S}, Act) \geq \theta_{s_i} \ \wedge \\ size(\mathbb{S}) \leq 20 \ \wedge \quad & sup(\mathbb{S}, InAct) \leq \theta_{s_a} \end{aligned} \tag{1.3.10}$$

Here, the expert is looking for pattern sets that are frequent in the actives, infrequent in the inactives, contain pattern p_1 and have size at most 20.

Fourth, one may want to find sets of patterns that cover similar sets of examples in Act by employing query Q_4.

$$min(red(\mathbb{S}, Act)) \geq 5 \wedge size(\mathbb{S}) \geq 2 \tag{1.3.11}$$

This requires that the intersection of each pair of patterns covers at least 5 examples and that the size of the pattern set is larger than 2. This could be used to identify families of patterns that characterize the same instances – i.e. equivalence classes from each of which one pattern is selected and the rest discarded.

Finally, an associative classification rule learner such as CBA aims at selecting a subset of the mined rules with high accuracy as the final classifier. The rule set should have high accuracy on a validation set and little redundancy among the rules. Thus Q_5 has the form:

$$\max(red(\mathbb{S}, D)) \leq \theta_{red} \wedge \arg_1 \max acc(\mathbb{S}, \mathcal{D}_{val}) \geq \theta_{acc} \tag{1.3.12}$$

The second term denotes that we are querying for the single most accurate pattern set among those with accuracy at least θ_{acc}.

1.3.5 Properties of constraints

Constraint based data mining system rely heavily on the properties of the employed constraints in order to 'push' the constraints into the data mining system and to develop efficient and effective algorithms. Therefore, since our aim is the formulation of a framework for constraint-based pattern set mining, we study in this section the properties of the pattern set constraint primitives introduced earlier. Many of the properties mirror those of constrained local pattern mining while some take the different nature of pattern sets into account.

Definition 1.3.17. Monotonicity *A constraint $\mathcal{C}$ is said to be* monotone *w.r.t. a generality relation $\preceq$ if and only if*

$$\forall \mathbb{G}, \mathbb{S}, \mathbb{G} \preceq \mathbb{S} : \mathcal{C}(\mathbb{G}) \Rightarrow \mathcal{C}(\mathbb{S}).$$

It is said to be anti-monotone *if and only if*

$$\forall \mathbb{G}, \mathbb{S}, \mathbb{G} \preceq \mathbb{S} : \mathcal{C}(\mathbb{S}) \Rightarrow \mathcal{C}(\mathbb{G}).$$

This definition is applicable to pattern set mining using the relations $\preceq_s$ or $\preceq_\subseteq$. When the monotonicity property holds at the pattern level set for $\preceq_\subseteq$ but not for $\preceq_s$, we say that the constraint is *restricted* monotone. Restricted anti-monotone is defined similarly.

To give an intuition of the meaning of this property, monotone formalizes the notion that a constraint that is satisfied will stay satisfied if the pattern set becomes more specific. Similarly, anti-monotone makes the same statement about a pattern set that becomes more general.

Constraints 1.3.2, 1.3.5, and 1.3.6, are give examples of anti-monotone constraints, while the remaining three (1.3.3, 1.3.4, 1.3.7) are monotone.

It has been shown for the local pattern mining setting that one can logically combine monotone and anti-monotone predicates. If the a's are anti-monotone and the m's monotone, then

- $\neg a$ is monotone and $\neg m$ is anti-monotone;
- $a_1 \wedge a_2$ and $a_1 \vee a_2$ are anti-monotone; and
- $m_1 \wedge m_2$ and $m_1 \vee m_2$ are monotone.

A relaxation of the monotone and anti-monotone properties that is still useful for local pattern mining assumes that a particular lexicographic order on the patterns exists. One then often talks about *convertible* constraints. To define convertible constraints, we will – for simplicity – assume that the generality relation corresponds to the subset relation.

Definition 1.3.18. Convertible Monotonicity *A constraint $\mathcal{C}$ is said to be* convertible anti-monotone *(resp. convertible* monotone*) w.r.t. a pattern language $\mathcal{L}_p$, if and only if there exists an order $\trianglelefteq$ on $\mathcal{L}_p$, such that for all ordered pattern-sets $\{p_1, \ldots, p_n\}$(with $p_1 \trianglelefteq \ldots \trianglelefteq p_n$):*

$$\mathcal{C}(\{p_1, \ldots, p_{n-1}\}) \Rightarrow \mathcal{C}(\{p_1, \ldots, p_n\})$$

$$(resp.\ \mathcal{C}(\{p_1, \ldots, p_n\}) \Rightarrow c(\{p_1, \ldots, p_{n-1}\})).$$

Example 1.3.4. *Two examples of convertible anti-monotone constraints are the ones used in* CBA*, discussed in Section 4.2, and in our* Bouncer *and* Picker *algorithms which we describe in Section 3.2. Both of those constraints make use of the orders imposed on the respective pattern sets:*

$$\mathcal{C}_{CBA}(\mathbb{S}, \mathcal{D}) \equiv \forall p \Rightarrow q \in \mathbb{S}, \exists e \in \mathcal{D} : e \notin \bigcup_{p_i \trianglelefteq_{CBA} p} cov(p_i, \mathcal{D}) \wedge class(e) = q$$

The CBA*-constraint states that each class-association rule in the pattern set must cover at least one instance that has not been covered by a predecessor pattern according to $\trianglelefteq_{CBA}$, and that it has to predict the instance's class correctly.*

$$\mathcal{C}_{Bouncer}(\mathbb{S}, \mathcal{D}) \equiv \forall p \in \mathbb{S} : \Phi(\bigcup_{p_i \trianglelefteq_B p}, p) \geq \theta$$

The BOUNCER*-constraint sets a minimum threshold on a measure applied to a pattern and the union of all predecessor patterns already in the pattern set. If the respective orders are used in the enumeration process, convertible monotonicity is ensured.*

A third type of constraint that is often used is that of succinctness (Ng et al. 1998). We again use the simpler definition already employed in the context of local pattern mining constraints:

Definition 1.3.19. Succinctness *A constraint predicate $\mathcal{C}$ defined on pattern sets is* succinct *if and only if for all sets $\mathbb{S}$: $\mathcal{C}(\mathbb{S})$ can be expressed as $\forall p \in \mathbb{S} : r(p) = true$ for a predicate r.*

Succinct constraints are attractive because they allow one to test whether a potential solution satisfies a constraint by ensuring that all members satisfy the constraint. This can be used to restrict the alphabet from which solutions can be assembled.

Example 1.3.5. *In a post-processing setting, where $\mathbf{L} = Th(\mathcal{L}_p, \mathcal{D}, c)$ has already been mined, a straight-forward succinct pattern set constraint is the minimum support constraint* $\min(sup((\mathbb{S}, \mathcal{D})) \geq \theta_{sup}$*. It can be enforced by restricting the set of patterns from which to assemble sets to $\forall p \in \mathbf{L} : sup(p, \mathcal{D}) \geq \theta_{sup}$.*

The boundable definition we used before (Definition 1.1.18) holds for the domain of pattern sets as well

Definition 1.3.20. Boundable *A function $f : \mathbf{L} \mapsto \mathbb{R}$ is said to be* upper-boundable (resp. lower-boundable) *with regard to a pattern set language $\mathbf{L}$ if given $f(\mathbb{S}) = k$, $k' \in \mathbb{R}$ can be derived s.t. $f(\mathbb{S}') \leq k'$ (resp. $f(\mathbb{S}') \geq k'$) for all $\mathbb{S}' \prec \mathbb{S}$ or $\mathbb{S} \prec \mathbb{S}'$.*

Contrary to local pattern mining, in pattern set mining both generalization and specialization bounding is possible, due to the different notion of matching.

Theorem 1.3.1. *Each constraint mentioned in Table 1.2 possesses the properties listed.*

Proof argumentation Most of the claims directly follow from the properties at the level of local patterns, and the duality w.r.t. the generalization relation. Therefore, we do not provide a formal proof. Nevertheless, some of the more interesting constraints are marked with a $'\star'$, resp. $'+'$, or $'\#'$.

Constraints marked with a $\star$, $\min(red(\mathbb{S}, \mathcal{D})) \leq \theta$ and $\min(red(\mathbb{S}, \mathcal{D})) \leq \theta$, are *restricted anti-monotone* and *restricted monotone*, respectively. The justification is that adding a pattern to a pattern set potentially decreases the minimal redundancy any two patterns can have, while removing a pattern potentially increases it. We first prove the anti-monotonicity of $\min(red(\mathbb{S}, \mathcal{D})) \leq \theta$ for subset generalization:

Theorem 1.3.2. *The minimum redundancy constraint* $\min(red(\mathbb{S}, \mathcal{D})) \leq \theta$ *is anti-monotone under subset refinement.*

Table 1.2: Set constraints and their properties.

Constraint	Property	Type
$G \preceq \mathbb{S}$	monotone	primitive
$\mathbb{S} \preceq S$	anti-monotone	primitive
$size(\mathbb{S}) \geq \theta$	anti-monotone	primitive
$size(\mathbb{S}) \leq \theta$	monotone	primitive
$sup(\mathbb{S}, \mathcal{D}) \leq \theta$	monotone	primitive
$sup(\mathbb{S}, \mathcal{D}) \geq \theta$	anti-monotone	primitive
$rep(\mathbb{S}, D_1, \mathcal{D}) \leq \theta$	generalization boundable$^{\#}$	primitive
$rep(\mathbb{S}, D_1, \mathcal{D}) \geq \theta$	generalization boundable$^{\#}$	primitive
$all(rep(\mathbb{S}, D_1, \mathcal{D})) \leq \theta$	succinct	aggregate
$all(rep(\mathbb{S}, D_1, \mathcal{D})) \geq \theta$	succinct	aggregate
$max(sup(\mathbb{S}, \mathcal{D})) \leq \theta$	succinct	aggregate
$min(sup(\mathbb{S}, \mathcal{D})) \geq \theta$	succinct	aggregate
$max(red(\mathbb{S}, \mathcal{D})) \leq \theta$	monotone	aggregate
$max(red(\mathbb{S}, \mathcal{D})) \geq \theta$	anti-monotone	aggregate
$min(red(\mathbb{S}, \mathcal{D})) \leq \theta$	restricted anti-monotone*	aggregate
$min(red(\mathbb{S}, \mathcal{D})) \geq \theta$	restricted monotone*	aggregate
$avg(sup(\mathbb{S}, \mathcal{D}))\{\geq, \leq\}\theta$	convertible	aggregate
$max(rep(\mathbb{S}, D_1, \mathcal{D})) \leq \theta$	succinct	aggregate
$min(rep(\mathbb{S}, D_1, \mathcal{D})) \geq \theta$	succinct	aggregate
$sum(red(\mathbb{S}, \mathcal{D})) \leq \theta$	monotone	aggregate
$sum(red(\mathbb{S}, \mathcal{D})) \geq \theta$	anti-monotone	aggregate
$\chi^2(sup(\mathbb{S}, D_+), sup(\mathbb{S}, D_-)) \geq \theta$	specialization upper-boundable	primitive
$min(dist(\mathbb{S}, \mathcal{D})) \geq \theta$	restricted monotone^{+}	aggregate
$min(dist(\mathbb{S}, \mathcal{D})) \leq \theta$	restricted anti-monotone^{+}	aggregate
$max(dist(\mathbb{S}, \mathcal{D})) \leq \theta$	restricted monotone^{+}	aggregate
$max(dist(\mathbb{S}, \mathcal{D})) \geq \theta$	restricted anti-monotone^{+}	aggregate

Proof. We want to prove that $min(red(\mathbb{S}, \mathcal{D})) \leq \theta_r \Rightarrow min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) \leq \theta_r$ or, reformulated, that $min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) \leq min(red(\mathbb{S}, \mathcal{D})$. Given $\mathbb{S} = \{p_1, \ldots, p_m\}$ then

$$min(red(\mathbb{S}, \mathcal{D})) = \min_{\forall p_i, p_j \in \mathbb{S}, p_i \neq p_j} red(p_i, p_j, \mathcal{D})$$

Adding an additional pattern p_{m+1} to the set leads to case 1:

$$\forall p_i \in \mathbb{S} : red(p_i, p_{m+1}, \mathcal{D}) \geq \min_{p_j \in \mathbb{S}, p_j \neq p_i} red(p_i, p_j, \mathcal{D})$$

which is equivalent to

$$min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) = min(red(\mathbb{S}, \mathcal{D})) \tag{1.3.13}$$

case 2:

$$\exists p'_i \in \mathbb{S} : red(p'_i, p_{m+1}, \mathcal{D}) < \min_{p_j \in \mathbb{S}, p_j \neq p'_i} red(p'_i, p_j, \mathcal{D}) \wedge$$
$$red(p'_i, p_{m+1}, \mathcal{D}) \geq min(red(\mathbb{S}, \mathcal{D}))$$

which is equivalent to

$$min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) = min(red(\mathbb{S}, \mathcal{D})) \tag{1.3.14}$$

or case 3

$$\exists p'_i \in \mathbb{S} : red(p'_i, p_{m+1}, \mathcal{D}) < \min_{p_j \in \mathbb{S}, p_j \neq p'_i} red(p'_i, p_j, \mathcal{D}) \wedge$$
$$red(p'_i, p_{m+1}, \mathcal{D}) < min(red(\mathbb{S}, \mathcal{D}))$$

equivalent to

$$min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) < min(red(\mathbb{S}, \mathcal{D})) \tag{1.3.15}$$

The combination of 1.3.13, 1.3.14, and 1.3.15 leads to

$$min(red(\mathbb{S} \cup \{p_{m+1}\}, \mathcal{D})) \leq min(red(\mathbb{S}, \mathcal{D}),$$

with a analogous argumentation holding for the case of specialization by pattern removal. □

If syntactic generalization or specialization is used, the properties turn into their dual, as we will prove for syntactic generalization:

Theorem 1.3.3. *The minimum redundancy constraint* $\min(red(\mathbb{S}, \mathcal{D})) \leq \theta$ *is monotone under syntactic refinement.*

Proof. We want to prove that under syntactic generalization

$$\forall p_s \in \mathbb{S} : min(red(\mathbb{S}, \mathcal{D})) \geq \theta_r \Rightarrow min(red(\mathbb{S} \cup \{p_g\} \setminus \{p_s\}, \mathcal{D})) \geq \theta_r$$

or, reformulated, that

$$\forall p_s \in \mathbb{S} : min(red(\mathbb{S} \cup \{p_g\} \setminus \{p_s\}, \mathcal{D})) \geq min(red(\mathbb{S}, \mathcal{D})).$$

Given $\mathbb{S} = \{p_1, \ldots, p_m\}$ then

$$min(red(\mathbb{S}, \mathcal{D})) = \min_{\forall p_i, p_j \in \mathbb{S}, p_i \neq p_j} red(p_i, p_j, \mathcal{D})$$

Let us assume without loss of generality that $p_i = p_1$ and $p_j = p_2$

$$red(p_1, p_2, \mathcal{D}) = |cov(p_1, \mathcal{D}) \cap cov(p_2, \mathcal{D})|$$

If $p_s \notin \{p_1, p_2\}$ then we have that

$$min(red(\mathbb{S} \cup \{p_g\} \setminus \{p_s\}, \mathcal{D})) = min(red(\mathbb{S}, \mathcal{D})) \tag{1.3.16}$$

If $p_s \in \{p_1, p_2\}$, let us say $p_s = p_1$ then

$$cov(p_g) \supseteq cov(p_1) \Rightarrow |cov(p_g)| \geq |cov(p_1)| \Rightarrow red(p_g, p_2, \mathcal{D}) \geq red(p_1, p_2, \mathcal{D})$$

Which leads to

$$min(red(\mathbb{S} \cup \{p_g\} \setminus \{p_s\}, \mathcal{D})) \geq min(red(\mathbb{S}, \mathcal{D})) \tag{1.3.17}$$

1.3.16 and 1.3.17 lead to

$$min(red(\mathbb{S} \cup \{p_g\} \setminus \{p_s\}, \mathcal{D})) \geq min(red(\mathbb{S}, \mathcal{D})).$$

Again, an analogous argumentation holds for the case of generalization. □

Since $min(red(\mathbb{S}, \mathcal{D})) \leq \theta$ ($min(red(\mathbb{S}, \mathcal{D})) \geq \theta$) is anti-monotone (monotone) for subset refinement (Theorem 1.3.2) and monotone (anti-monotone) for syntactic refinement (Theorem 1.3.3), it is restricted anti-monotone (restricted monotone).

For those constraints with a $'+'$, $max(dist(\mathbb{S}, \mathcal{D})) \geq \theta$ and $min(dist(\mathbb{S}, \mathcal{D})) \leq \theta$ are restricted anti-monotone since subset generalization increases the number of possible pairings. It might increase the largest distinctiveness that two patterns in this set show and decrease the smallest distinctiveness. Subset specialization has the opposite effects on the maximal and minimal values of distinctiveness, making $max(dist(\mathbb{S}, \mathcal{D})) \leq \theta$ and $min(dist(\mathbb{S}, \mathcal{D})) \geq \theta$ restricted monotone. We omit the formal proof of these properties since it consists of arguments similar to the proof for redundancy-properties.

Under syntactic refinement, the maximum constraints $max(dist(\mathbb{S}, \mathcal{D}))\{\leq, \geq\}\theta$ and the minimum constraints $min(dist(\mathbb{S}, \mathcal{D}))\{\leq, \geq\}\theta$ become boundable. For these constraints to be boundable, $\forall p_i, p_j \in \mathbb{S} : dist(p_i, p_j, \mathcal{D})$ must be boundable. We prove this for the case of syntactic specialization:

Theorem 1.3.4. *Given $p_i, p_j, dist(p_j, p_i, \mathcal{D})$, $dist(p_j, p_s, \mathcal{D})$ is boundable under syntactic specialization, that is, it is possible to calculate a lower and an upper bound $ub_l \leq dist(p_j, p_s, \mathcal{D}) \leq ub_s$.*

Proof. Given $\mathbb{S} = \{p_1, \ldots, p_m\}$, if any p_i is selected for specialization to p_s, we have that:

$$cov(p_i, \mathcal{D}) \subseteq cov(p_s, \mathcal{D}) \Rightarrow |cov(p_i, \mathcal{D})| \leq |cov(p_s, \mathcal{D})|$$

In the case $cov(p_i, \mathcal{D}) = cov(p_s, \mathcal{D})$ we have that

$$\forall p_j \in \mathbb{S}, p_j \neq p_i dist(p_j, p_i, \mathcal{D}) = dist(p_j, p_s, \mathcal{D})) \tag{1.3.18}$$

Recalling the definition of distinctiveness:

$$\text{dist}(p_i, p_j, \mathcal{D}) = \text{diff}(p_i, p_j, \mathcal{D}) = \{cov(p_1, \mathcal{D}) \cup cov(p_2, \mathcal{D})\} \setminus ovlp(p_1, p_2, \mathcal{D})$$

and the definition of overlap:

$$ovlp(p_1, p_2, \mathcal{D}) = cov(p_1, \mathcal{D}) \cap cov(p_2, \mathcal{D})$$

three properties hold :

$$1)\forall p_j \in \mathbb{S}, p_j \neq p_i : \{\} \subseteq ovlp(p_j, p_s, \mathcal{D}) \subseteq ovlp(p_j, p_i, \mathcal{D}) \tag{1.3.19}$$

Which gives us bounds for the overlap $ovlp(p_j, p_s, \mathcal{D})$

$$\begin{aligned} &2)\forall e \in ovlp(p_j, p_i, \mathcal{D}) \Rightarrow e \in cov(p_j, \mathcal{D}) \wedge e \in cov(p_i, \mathcal{D}) \\ &\Leftrightarrow \forall e \in ovlp(p_j, p_i, \mathcal{D}), e \notin ovlp(p_j, p_s, \mathcal{D}) \\ &\Rightarrow e \in cov(p_j, \mathcal{D}), e \notin cov(p_s, \mathcal{D}) \Rightarrow e \in cov(p_j, \mathcal{D}) \cup cov(p_s, \mathcal{D}) \end{aligned} \tag{1.3.20}$$

which means that for each instance that is removed from the overlap due to specialization, the union of coverages stays the same

$$3)\forall e \in cov(p_i, \mathcal{D}), e \notin ovlp(p_i, p_j, \mathcal{D}), e \notin cov(p_s, \mathcal{D}) \Rightarrow e \notin cov(p_j, \mathcal{D}) \cup cov(p_s, \mathcal{D}) \tag{1.3.21}$$

which means that for each instance that was not in the overlap, yet is removed from the coverage of p_s due to specialization, the union of coverages shrinks

We can combine the lower bound of 1.3.19 and 1.3.20 to formulate an upper bound:

$$dist(p_j, p_s, \mathcal{D}) = |diff(p_j, p_s, \mathcal{D})| \leq |cov(p_j, \mathcal{D}) \cup \{cov(p_i, \mathcal{D}) \setminus ovlp(p_j, p_i, \mathcal{D})\}| \tag{1.3.22}$$

and the upper bound of 1.3.19 and 1.3.21 to formulate a lower bound:

$$dist(p_j, p_s, \mathcal{D}) = |diff(p_j, p_s, \mathcal{D})| \geq |cov(p_j, \mathcal{D}) \setminus ovlp(p_j, p_i, \mathcal{D})| \tag{1.3.23}$$

Combining 1.3.22 and 1.3.23 means that $max(dist(\mathbb{S}, \mathcal{D}))$ and $min(dist(\mathbb{S}, \mathcal{D}))$ become boundable measures and the according constraints boundable constraints. □

For the representativeness marked with a $'\#'$, let us illustrate this using an example.

Example 1.3.6. *Assume that the underlying dataset consists of two subsets of size 300 each. If a pattern set* $\mathbb{S}$ *covers 200 instances of dataset one and 100 instances of dataset two,* $rep(\mathbb{S}, D_1, \mathcal{D}) = \frac{2}{3}$*. Generalizing this set to a set* $\mathbb{S}'$ *will increase either* $sup(\mathbb{S}', D_1)$*,* $sup(\mathbb{S}', D_2)$*, or both. Thus* $0.4 \leq rep(\mathbb{S}', D_1, \mathcal{D}) \leq 0.75$*, representativeness is generalization boundable.*

Using this list of properties, we can now investigate the properties of the conjunctive queries, given earlier as examples:

- Q_1 is written as the conjunction of an anti-monotone and an monotone constraint, i.e., in the form 'anti-monotone $\wedge$ monotone';
- Q_2 is in the form 'succinct $\wedge$ monotone $\wedge$ anti-monotone';
- Q_3 is in the form (anti-monotone $\wedge$ anti-monotone) $\wedge$ (monotone $\wedge$ monotone)', and hence, 'anti-monotone $\wedge$ monotone';
- Q_4 is in the form 'restricted monotone $\wedge$ anti-monotone; and
- Q_5 is monotone $\wedge$ 'generalization upper boundable'.

Observe that it is – of course – also possible to provide further queries and analyze their properties. For instance, the specific type of query which was answered by (Shima et al. 2005) corresponds to

$$|\bigcup_{i<j, p_i, p_j \in \mathbb{S}} ovlp(\{p_i, p_j\}, \mathcal{D})| \leq \theta. \tag{1.3.24}$$

It is a monotone constraint.

1.4 Summary

In this chapter of our work we laid out the foundations needed for local pattern mining and pattern set mining. Additionally, we explained the motivation behind pattern set mining: patterns are usually needed for addressing KDD tasks and pattern sets define the ways in which those patterns are assembled for use. This can include the selection of subsets of patterns as features for the description of data to model formation out of predictive rules.

There are many similarities among the two tasks which were mirrored in the layout of the two sections (Section 1.1 and 1.3). In both cases, we started by discussing how to express the results we aim to mine. From these formulations follow the ways that they are applicable to any data which in turn spawn several properties that hold no matter the data considered. Particular data effects the so-called a posteriori properties. The interestingness of patterns and pattern sets lies very much in the eye of the beholder, the user in the case of data mining. To define this interestingness, constraints are used in both cases which can then be used to make the search for solutions more effective.

These similarities should not obscure the biggest difference between Sections 1.1 and 1.3: While local pattern mining is well-established and a formal framework exists, in which one can reason about properties of languages, measures, and constraints, such a principled way of talking about pattern set mining has been absent from the literature so far. Having such a framework is useful in terms of developing algorithms for finding good solutions efficiently. In contrast to this, pattern set mining techniques in the literature are mainly tailored to specific KDD tasks, or focussing on either improving only effectiveness or efficiency. We will describe existing approaches to local pattern mining and pattern set mining in the next chapter and discuss how one can move from the data and a general idea of the purpose of a pattern set to formulating a way to approach the task. Most importantly, we introduce two novel algorithms, inspired by exhaustive local pattern mining, and feature selection methods from machine learning.

Chapter 2

Discussing Algorithms

When writing about finding the theories of local patterns $Th(\mathcal{L}_p, \mathcal{D}, c)$ and of pattern sets $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$ so far, we only have discussed the theoretical foundations but not yet the practical approaches towards finding them. Since there cannot be an algorithmic solution that is well-suited to *all* problem settings, we will discuss several options in this chapter. Data mining is essentially search and the main trade-off is between completeness and efficiency of an algorithm:

- While an exhaustive technique is usually *guaranteed* to find the best or the complete solution, it often pays a high computational cost (extensive running times and/or high memory consumption).

- A heuristic technique, on the other hand, can be expected to finish *quickly* yet does not necessarily return the best solution and will not be able to return a complete solution without deteriorating to exhaustive search.

Algorithmically, approaches differ in how they traverse the space of potential solutions. As an illustration, consider the lattice of itemsets shown as Figure 2.1. All of these are potential solutions with regard to constraint satisfaction or the optimization (maximization/minimization) of an interestingness measure. Both in the case of exhaustive search (Section 2.1) and heuristic search (Section 2.2), a combination of selection of partial solutions and of the refinement operator used determines which parts of this solution space are traversed. After explaining both approaches in a general way, we will discuss several concrete algorithms for local pattern mining and pattern set mining. Common to all these techniques is that refinement operators work by only adding/removing or specializing/generalizing patterns.

The third section (Section 2.3) differs from the other two in that moving from one partial solution to the next involves more elaborate refinement operators that manipulate the underlying data. We will explain three ways for manipulating data based on the partial local pattern or partial pattern set and present several concrete techniques. While it is possible in theory to use this

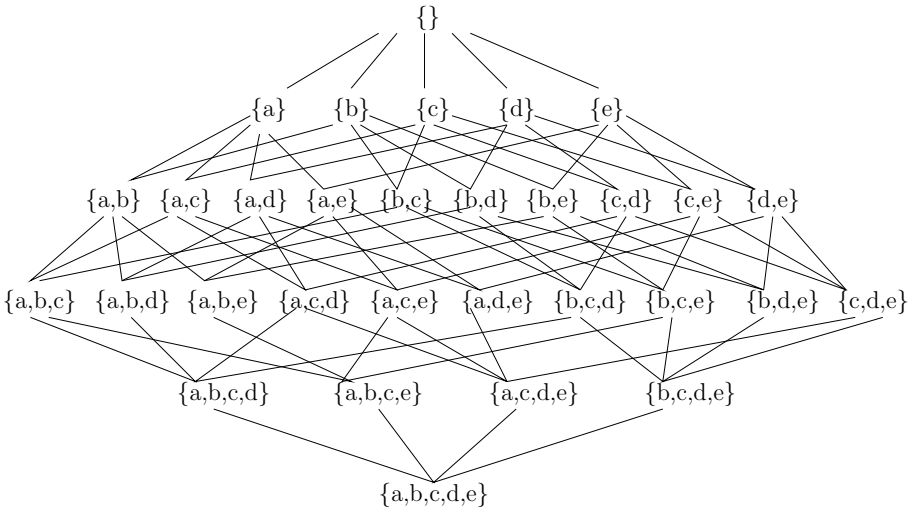


Figure 2.1: Complete lattice over the pattern space for items a, b, c, d, e

way of iterative mining both exhaustively and heuristically, we will argue why the exhaustive case is too expensive.

As we have explained earlier, both the data mining and machine learning community have somewhat of a blind spot when it comes to the employment of the algorithmic solutions that have been developed in their respective fields. In data mining this means that general exhaustive techniques that have been developed for local pattern mining have yet to see application to the problem of pattern set mining, a first attempt was made in (De Raedt and Zimmermann 2007). Our contribution in this regard is the discussion of mining in general terms, which together with the framework for pattern set mining, makes it clear that such techniques can be adapted rather easily. The machine learning community, on the other hand, has mainly developed systems in which the local pattern mining and pattern set mining step are integrated and the entire system is viewed as monolithic. While there has been literature that discussed certain pattern set mining (or induction) schemes in general terms, for instance (Fürnkranz and Flach 2005), typically the integration of the pattern set mining step with a (heuristic) local pattern mining step is taken for granted. We take care to discuss that those are distinct, although interacting, issues, opening up a wider range of algorithmic combinations. In our view, pattern set mining can in general be understood in terms of a "wrapper" approach of which local pattern mining is one step.

We will present algorithms in general search terms, referring to *solutions* instead of patterns or pattern sets. Those solutions are formulated in a language $\mathcal{L}$. It can be instantiated by a pattern language like the ones defined in Section 1.1, or by pattern set languages, such as defined in Section 1.3.1. In both types of mining – local pattern mining and pattern set mining – usually only

one "direction" of refinement is employed, that is, either a generalization or a specialization operator is used. We will, for illustrative purposes, assume that refinement starts from the empty set, meaning that a specialization operator is used for local pattern mining and a generalization operator for pattern set mining.

2.1 Exhaustive Search

Exhaustive miners needs to traverse the entire space of *potential* solutions during the mining operation. This means that it is necessary for the refinement operator to be *complete* (Definition 1.1.9), because otherwise solutions will be missed. It is also often attractive to use a refinement operator which is *optimal*, to avoid the enumeration (and evaluation) of duplicates.

Search through the lattice can be performed:

1. Breadth-first: in each step, *all* potential solutions at a given level in the lattice are refined

2. Depth-first: starting from the left, in each step the *deepest* potential solution in the lattice is refined until refinement is not possible anymore, then the process is continued with the second-deepest

3. Best-first: in each step, the *highest-valued* potential solution according to a heuristic quantifying the future quality is refined

In the first case, it is often desirable for reasons of computational efficiency to use a refinement operator which is *optimal* in addition to complete, to avoid the enumeration of duplicate solutions. If depth-first or best-first search is used for top-k mining, on the other hand, an *optimal* refinement operator *can* lead to a larger search space, if solutions are enumerated that do not increase the threshold quickly.

2.1.1 Complete mining

There are two tasks that have to be distinguished. The first one, which we will discuss in this section, is that of complete mining which means that *all* patterns or pattern sets that satisfy particular constraints are returned at the end of the mining operation. Algorithm 1 is the general level-wise (that is, breadth-first) algorithm for local pattern and pattern set mining. Adaptation to depth-first search is straight-forward.

Note that the constraint to be satisfied can have any properties for this algorithm to work, since only satisfaction is tested against. An algorithm performing search like this will in fact enumerate the *entire* pattern space, leading to potentially extreme running times and potential exhaustion of working memory. Complete mining therefore *prunes* away pattern subspaces that cannot include solutions.

Algorithm 1 The general level-wise algorithm

Given: language $\mathcal{L}$, ideal refinement operator ρ, constraint c, data base $\mathcal{D}$
Return: set of all solutions $s \in \mathcal{L}$ satisfying c on $\mathcal{D}$: $Th(\mathcal{L}_p, \mathcal{D}, c)$ or $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

$\mathbb{S} = \emptyset, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  $\mathbb{S} = \mathbb{S} \cup \{s \mid s \in \mathbb{C} : c(s) = true\}$
  $\mathbb{P} = \mathbb{C}$
end while
return $\mathbb{S}$

For local patterns the most widely used type of constraint for pruning of level-wise search is that of anti-monotone constraints. The definition of anti-monotonicity (Definition 1.1.15) implies that for two patterns $g, s : g \prec s$, $c(g) = false \Rightarrow c(s) = false$. The best-known example is the APRIORI algorithm (Agrawal and Srikant 1994). Similarly, if the enumeration is performed from specific to general (as in pattern set mining), monotone constraints have the same pruning effect, as we showed in (De Raedt and Zimmermann 2007). The set of potential solutions $\mathbb{P}$ can therefore be reduced, improving the efficiency (see Algorithm 2).

Algorithm 2 The general level-wise algorithm with pruning

Given: language $\mathcal{L}$, ideal refinement operator ρ, (anti-)monotone constraint c, data base $\mathcal{D}$
Return: set of all solutions $s \in \mathcal{L}$ satisfying c on $\mathcal{D}$: $Th(\mathcal{L}_p, \mathcal{D}, c)$ or $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

$\mathbb{S} = \emptyset, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  $\mathbb{S} = \mathbb{S} \cup \{s \mid s \in \mathbb{C} : c(s) = true\}$
  $\mathbb{P} = \{s \mid s \in \mathbb{C} : c(s) = true\}$
end while
return $\mathbb{S}$

The goal might be to mine all patterns whose score, as determined by a correlation measure, exceeds a certain minimum threshold, however. Correlation measures are usually not anti-monotone and therefore do not allow the type of pruning shown in Algorithm 2. If the correlation measure is *boundable* (Definition 1.1.18), upper-bound pruning is possible. In the next section, we introduce coverage spaces to facilitate the explanation of upper-bound pruning and contrast it with anti-monotone pruning.

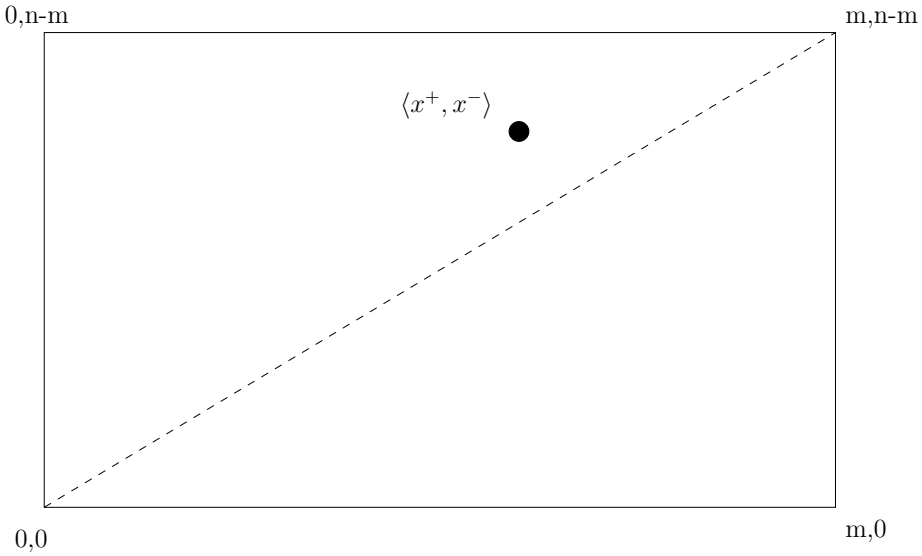


Figure 2.2: General coverage space

Coverage spaces, anti-monotone pruning and why correlation measures can be tricky

A coverage space takes the form shown in Figure 2.2. Connected to the contingency table we introduced in Section 1.1.2, and the definition of a stamp point (Definition 1.1.12), a coverage space is a diagram spanning all possible stamp points w.r.t. a target pattern q. Coverage spaces were first introduced and used in such a way by Flach *et al.* (2005).

Since a pattern can only be present or absent, i.e. is binary, the coverage space is a two-dimensional diagram. We adopt the convention that the vertical axis denotes the number of instances where q is absent, the horizontal one the number of instances where q is present. The lower left corner corresponds to the stamp point $\langle 0, 0 \rangle$, the *sp* of a pattern that does not cover any instance in the current data set. The upper left corner is $\langle 0, n - m \rangle$, and the lower right $\langle m, 0 \rangle$, patterns covering *all* the instances where q is absent (present). Finally, the upper right is occupied by all patterns covering everything, with the stamp point $\langle m, n - m \rangle$. The dashed diagonal line connects all the stamp points that have the same distribution of positive and negative instances as the entire data set. Every pattern covering between 0 and n instances in the data can be located in this coverage space, such as the example pattern p shown there.

Let us now take a look at different quality measures of patterns using their isometrics. As we wrote in Section 1.1.2, any measure can be understood as a function $\sigma : \mathbb{N}^2 \mapsto \mathbb{R}$. So, although a faithful depiction of the function's shape would have to be done in a three-dimensional diagram, it is possible to project the shape of the function onto its domain, which is two-dimensional – the cover-

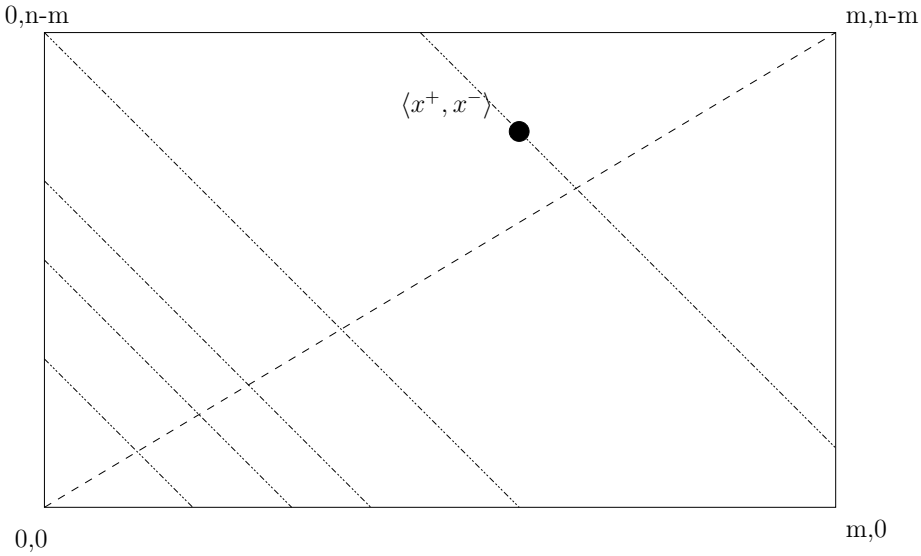


Figure 2.3: Support isometrics in coverage space

age space. More specifically, an isometric is a line connecting points in coverage space that evaluate to the same value of σ. Figure 2.3 shows several isometrics for the support- or frequency-isometric. As can be seen, support connects all stamp points that correspond to the same number of covered instances, no matter their distribution of the target pattern. Minimum-support mining selects all patterns with relatively high support, making the one residing on the upper right corner, covering all the data, the pattern most likely to be selected.

To visualize pruning with an anti-monotone constraint such as minimum support, consider Figure 2.4. It shows three stamp points of patterns $p_1 \prec p_2 \prec p_3$, and a support threshold isometric. p_1 has a support value that is larger than the threshold, while p_2 falls below the threshold. Due to the anti-monotonicity of the minimum support constraint, this information is enough to prune the entire patternsubspace $\rho^*(p_2)$ since compared to the threshold, the support of members of this subspace can only decrease.

Correlation measures take both support of a pattern into account and the difference from the background or standard distribution (denoted by the middle diagonal). Large differences in the distribution of the target pattern between the entire data and the patterns cover are rewarded, as is high support on the entire data. When visualized in coverage space, the further away from the diagonal, the more relevant and interesting is a pattern. If the measure is symmetric, e.g. χ^2 (see Example 1.1.2), isometrics are mirrored on the distribution diagonal (Figure 2.5), with the lower isometric achieving the same score as the upper.

If the measure is asymmetric, e.g. *WRAcc* (see Example 1.1.3), only patterns that occur *more often* together with the target pattern than the standard

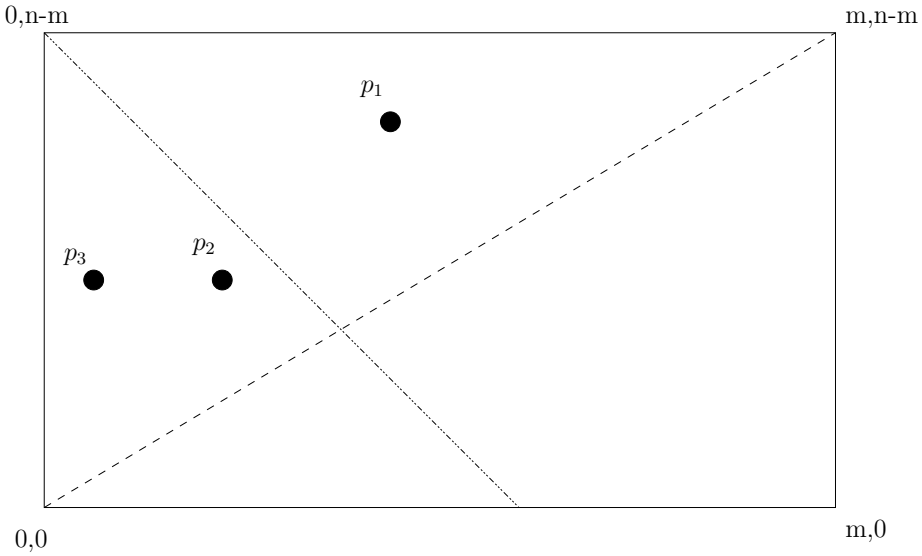


Figure 2.4: Support pruning in coverage space

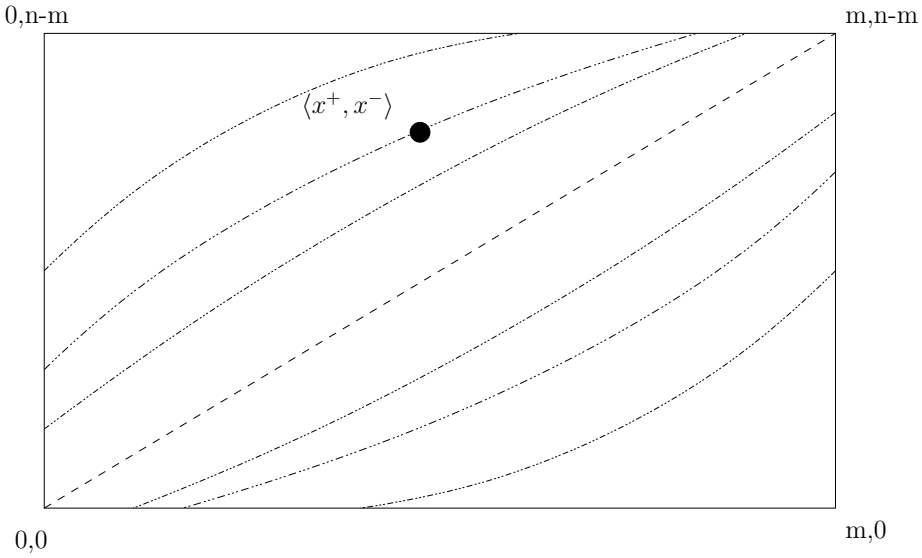


Figure 2.5: χ^2-isometrics in coverage space

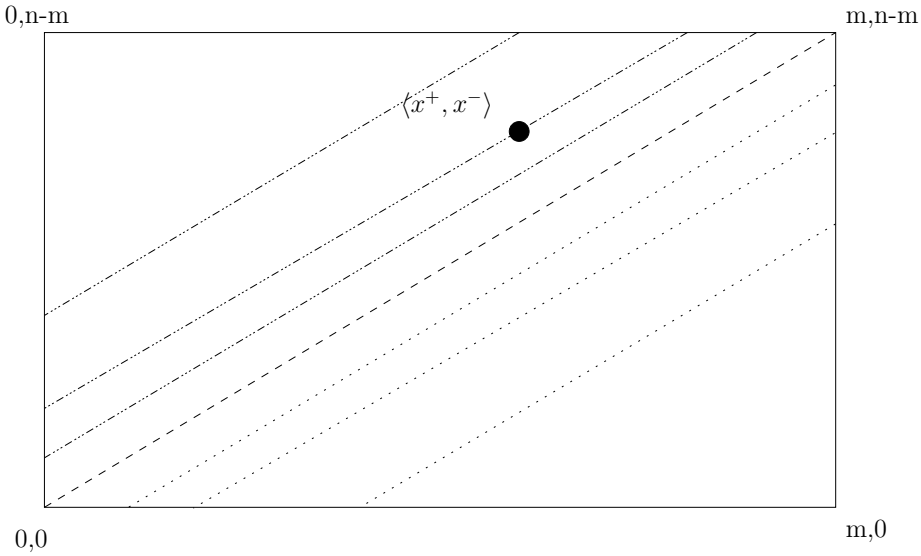


Figure 2.6: *WRAcc*-isometrics in coverage space

distribution, are given a positive score. Therefore, being above or below the diagonal leads to different scores and different isometrics as shown in Figure 2.6. To denote this, the isometrics in the figure are in different line styles above and below the diagonal, in contrast to isometrics for symmetric measures.

Due to the different shape and orientation of the isometrics simply pruning specializations of patterns that fall below the isometric does not work, as can be seen in Figure 2.7. We can see the same three stamp points and an isometric corresponding to a threshold of the χ^2 measure. As we noted above, farther away from the diagonal translates into higher scores and, in addition, χ^2 is symmetric, as can be seen from the shape of the threshold-isometric.

For measures whose isometrics are shaped like this, the anti-monotone property does not hold. This becomes clear when we have a look at the three stamp points - while the second one lies inside the thresholded area, meaning that this pattern falls below the threshold, p_3 exceeds it although it is a refinement of p_2. Obviously, a different pruning strategy is needed.

Convexity-based upper-bound pruning

Because the relationship $cov(p_1, \mathcal{D}) \geq cov(p_2, \mathcal{D}) \geq cov(p_3, \mathcal{D})$ still holds and since farther away from the diagonal translates into higher scores, the question that has to be asked for pruning is:

Given the information $sp(p_2) = \langle x^+, x^- \rangle$, is there is *any* potential stamp point $sp(p_3)$ whose value can exceed the threshold?

The question is a clear "yes"; quite a few actually: all those having values

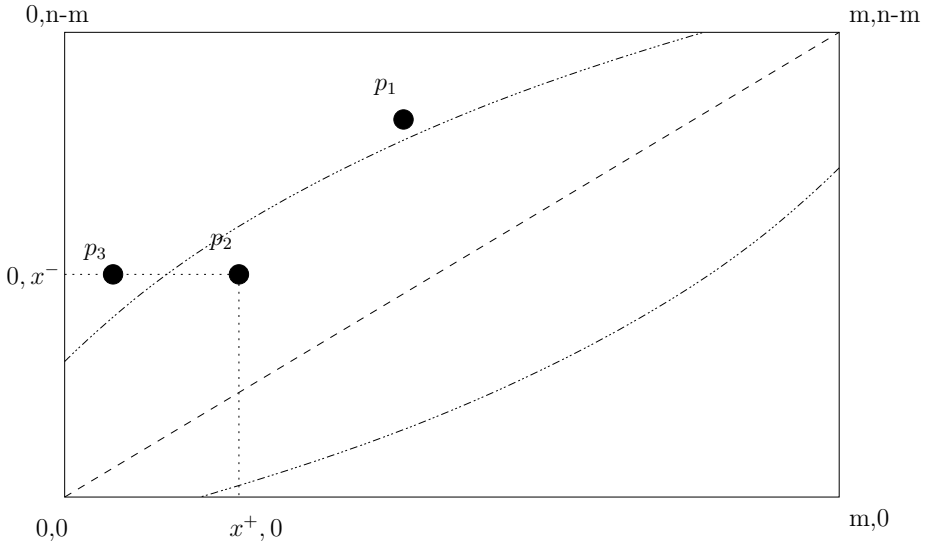


Figure 2.7: χ^2 pruning in coverage space

$\langle x^{+\prime}, x^{-\prime}\rangle : 0 \leq x^{+\prime} \leq x^+, 0 \leq x^{-\prime} \leq x^-$. Of this set, the two that are trivially furthest away from the diagonal lie at $\langle x^+, 0\rangle$ and $\langle 0, x^-\rangle$, of which the latter is the one scoring higher. Only if *neither* of those two stamp points can be evaluated to a value exceeding the threshold is it safe to prune a subspace originating at a pattern. Evaluating σ on those two extreme points gives us thus an upper bound on future values specializations of p_2:

$$\forall p' \in \rho^*(p_2) : \sigma(sp(p')) \leq \max\{\sigma(x^+, 0), \sigma(0, x^-)\}$$

There is, however, a caveat to this technique: it only works if the function values of all points on the line connecting $\langle x^+, x^-\rangle$ and any of the two extreme points lie below the line connecting the function values of the two stamp points. In fact, there must not be any stamp point within the area encompassed by the threshold isometrics whose function value lies above the line connecting the function values of two points between which it lies. Formulated, σ has to be convex:

Definition 2.1.1. Convexity *A function $\sigma : D \mapsto \mathbb{R}$ is convex if and only if $D \subseteq \mathbb{R}^d$ is a convex set and $\forall x_1, x_2 \in D, \lambda \in [0,1] : f(\lambda x_1 + (1-\lambda)x_2) \geq \lambda f(x_1) + (1-\lambda) f(x_2)$.*

If this is not the case, evaluating σ on $\langle x^+, 0\rangle$ and $\langle 0, x^-\rangle$ might underestimate the true value a specialization of the current pattern might attain, and more complex and therefore more time-consuming techniques would be needed to find the upper bound. Fortunately, a variety of correlation measures such as χ^2, *Information Gain*, *Weighted Relative Accuracy*, *Category Utility* (*CU*)

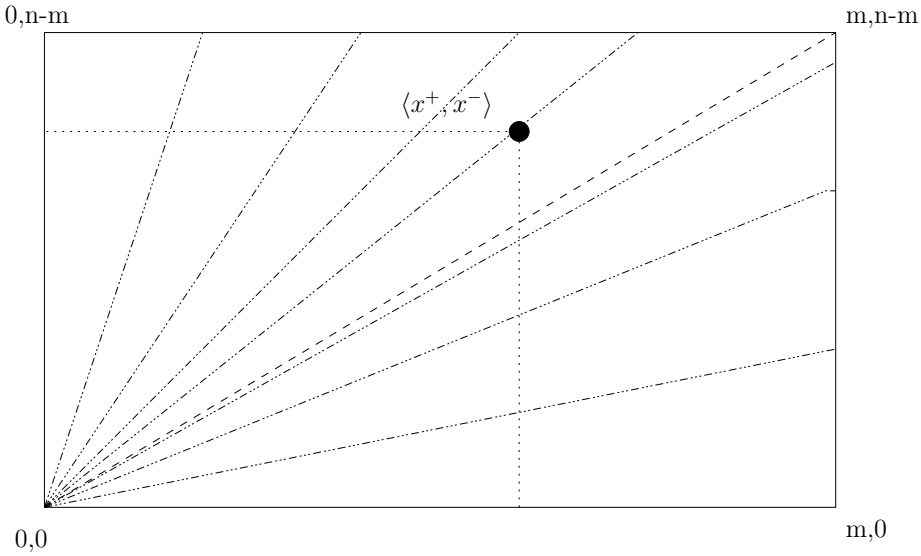


Figure 2.8: Confidence isometrics in coverage space

(which we introduce formally in Example 2.1.1) and others fulfill the second criterion. As to the first criterion, the convexity of the domain:

Definition 2.1.2. Convex Set *A set in Euclidean space $\mathbb{R}^d$ is convex set if it contains all the line segments connecting any pair of its points.*

A line between any two points inside or on the sides of a rectangle can never be outside the rectangle, which means that a rectangle is a convex set of points. Since the set of all stamp points can be visualized as a rectangular PN-space, the first condition of convexity is satisfied for the functions we consider in this work as well.

Theorem 2.1.1. *Given a pattern p, target pattern q, stamp point $sp(p) = \langle x^+, x^- \rangle$ and convex correlation measure σ, the upper bound ub_σ on $\sigma(sp(p)), p' \in \rho(p)$, is given by $ub_\sigma(p) = \max\{\sigma(x^+, 0), \sigma(0, x^-)\}$.*

Proof. The theorem can be proved by referring to the well-known fact that a convex function takes its maximal values on the extreme points of its domain. Given the domain borders x-axis, y-axis, and the lines connecting $sp(p)$ with those axes, candidates for those extreme values are the two points mentioned above, the origin $\langle 0, 0 \rangle$ and $sp(p)$ itself. Given that $\langle 0, 0 \rangle$ is also on the overall distribution diagonal, it cannot be a maximal value, and $sp(p)$ corresponds to a more specific pattern with the same performance as p, which means that no new information is added. □

Confidence (or accuracy in the context of machine learning) is also (trivially) convex, showing a different type of isometric (see Figure 2.8). Looking

at the isometrics, one sees that it radiates outward from the origin $\langle 0, 0\rangle$. Confidence clearly has a relation with the underlying distribution, as further from the diagonal is better. Unfortunately, it does not take into account *how many* instances are covered at all. The best patterns reside on the horizontal and vertical axes, corresponding to $conf(p, \mathcal{D}) = 1.0$ for predicting $q/\neg q$, respectively. As can be seen in the figure, any pattern's upper bound will always lie on those axes. Therefore, the upper bound is the same for any pattern, 1.0 or perfect confidence, making pruning impossible for a minimum threshold. This is the main reason why direct accuracy maximization is either a post-processing step to frequency-based mining, as in CBA or heuristic, as in CN2.

As stated before, we call functions for which non-trivial upper bounds can be calculated *boundable* (Definition 1.1.18). Boundable functions allow us to define boundable constraints, an example of which would be the local pattern mining constraint $c : \chi^2(sp(p), \mathcal{D}) \geq \theta_{sig}$. The upper bound allows pruning of pattern subspaces whose members cannot exceed the threshold. The level-wise algorithm for a minimum correlation score value takes the form shown as Algorithm 3.

Algorithm 3 The general level-wise algorithm with upper-bound pruning

Given: language $\mathcal{L}$, refinement operator ρ, boundable constraint $\sigma(sp(s)) \geq \theta_{sig}$, data base $\mathcal{D}$
Return: set of all solutions $s \in \mathcal{L}$ satisfying c on $\mathcal{D}$: $Th(\mathcal{L}, \mathcal{D}, c)$

$\mathbb{S} = \emptyset, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  $\mathbb{S} = \mathbb{S} \cup \{s \mid s \in \mathbb{C} : \sigma(sp(s)) \geq \theta_{sig}\}$
  $\mathbb{P} = \{s \mid s \in \mathbb{C} : sp(s) = \langle x^+, x^-\rangle, \sigma(\langle x^+, 0\rangle) \geq \theta_{sig} \vee \sigma(\langle 0, x^-\rangle) \geq \theta_{sig}\}$
end while
return $\mathbb{S}$

Extension of upper-bound pruning to several target patterns

Finally, we extend the upper-bound pruning technique for several target patterns. The description we gave above is with regard to a single target pattern, a useful technique for tasks such as classification and *subgroup discovery* (Lavrač et al. 2004). For *clustering* and the relatively new field of multi-target prediction (Vens et al. 2008), it is necessary to mine patterns that correlate with *several* independent patterns. We consider two target patterns independent if changes in the distribution of one of the two does not cause a change in the distribution of the other.

In a sense this is the worst-case scenario since the algorithm needs to consider the effect of the mined pattern on *all* target patterns, increasing computational complexity. If two target patterns really *are* interdependent then optimizing the change in distribution for one will have an effect on the other as well.

What this independence allows us to do is to treat the contingency tables for p and each of the q_i separately, keeping the low dimensionality and the notation of x_i^+, x_i^-, instead of using a proper multi-dimensional contingency table, whose dimension would rise with the number of patterns involved. We define an extended stamp point:

Definition 2.1.3. Extended Stamp Point *Given $\mathcal{D}$ and a set of fixed target patterns $\{q_1, \ldots, q_d\}$, the* extended stamp point *of p is*

$$sp_e(p) = \langle x_1^+, x_1^-, \ldots, x_d^+, x_d^- \rangle,$$

where $sp(p, q_i) = \langle x_i^+, x_i^- \rangle$

It should be noted that each q_i induces its own fixed m_i, that is for each q_i the number of instances in $\mathcal{D}$ in which it is present stays fixed throughout mining.

Due to the independent nature of target patterns, we can define a cumulative correlation measure:

Definition 2.1.4. Cumulative Correlation Measure *Given $\mathcal{D}$, a correlation measure $\sigma : \mathbb{N}^2 \mapsto \mathbb{R}$, and a set of fixed target patterns $\{q_1, \ldots, q_d\}$, the* cumulative *correlation measure $\sigma_{cum} : \mathbb{N}^{2d} \mapsto \mathbb{R}$ is defined as:*

$$\sigma_{cum}(sp_e(p)) = \sum_{i=1}^{d} \sigma(x_i^+, x_i^-).$$

An example of a cumulative correlation measure that has been used in the Cobweb algorithm (Fisher 1987), is *Category Utility*:

Example 2.1.1. *Given $\mathcal{D}$ and a set of fixed target patterns $\{q_1, \ldots, q_d\}$,* Category Utility *takes the form:*

$$\begin{aligned} CU(p, \mathcal{D}) &= CU(\langle x_1^+, x_1^-, \ldots, x_d^+, x_d^- \rangle) \\ &= \tfrac{1}{2} \sum_{i=1}^{d} \left(\tfrac{x_i^+ + x_i^-}{n} \left(\left(\tfrac{x_i^+}{x_i^+ + x_i^-} \right)^2 - \left(\tfrac{m_i}{n} \right)^2 + \left(\tfrac{x_i^-}{x_i^+ + x_i^-} \right)^2 - \left(\tfrac{n - m_i}{n} \right)^2 \right) \right. \\ &\quad \left. + \tfrac{n - (x_i^+ + x_i^-)}{n} \left(\left(\tfrac{m_i - x_i^+}{n - (x_i^+ + x_i^-)} \right)^2 - \left(\tfrac{m_i}{n} \right)^2 + \left(\tfrac{n - m_i - x_i^-}{n - (x_i^+ + x_i^-)} \right)^2 - \left(\tfrac{n - m_i}{n} \right)^2 \right) \right) \end{aligned}$$

Recalling our discussion of upper bounds before, this lends itself to a trivial upper bound $ub_{cum}(p) = \sum_{i=1}^{d} \max\{\sigma(x_i^+, 0), \sigma(0, x_i^-)\}$. Since each of the individual summands is an upper bound w.r.t. its target attribute, the entire sum is as well.

This upper bound is looser than it needs to be, however, as one can see considering Figure 2.9. It shows the coverage spaces for two different target patterns, overlaid over each other, with a pattern's stamp point in each of them, sharing the same support isometric. It also shows the points $\langle x_i^+, 0 \rangle$, and $\langle 0, x_i^- \rangle$, and the support isometrics that they induce. Considering those support isometrics, one recognizes that none of the extreme points lies on the isometric

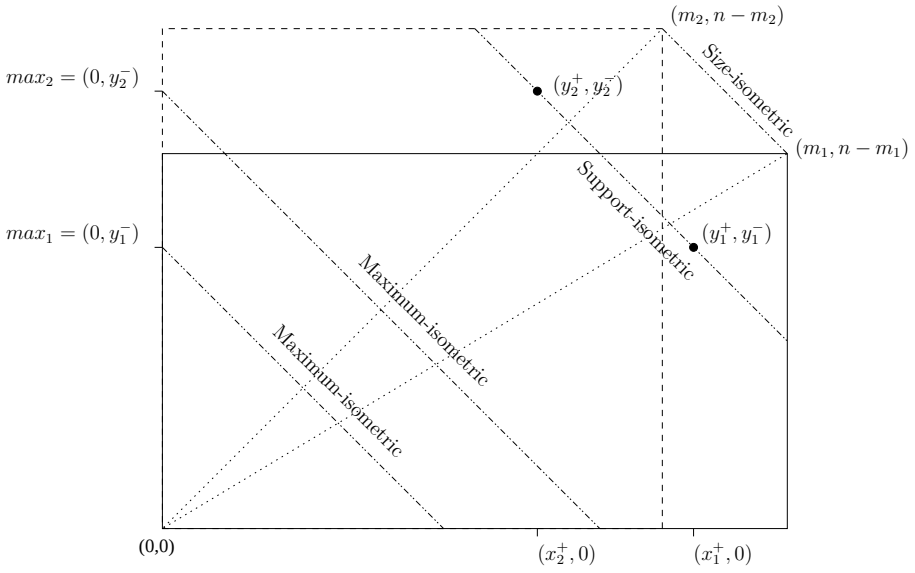


Figure 2.9: Inconsistency of naïve maxima

of one of the other extreme points. This in turn means that an upper bound that is calculated by simply adding up upper bounds for each target attribute quantifies an impossible future pattern – one that has different support on the data, depending on which target attribute one considers.

The solution to this is to only consider *possible* extreme points. Due to the anti-monotonocity of support, all possible support values of specializations of the current pattern are known:

$$p' \in \rho(p) : 0 \leq sup(p', \mathcal{D}), \leq sup(p, \mathcal{D})$$

For any possible future support value, it is possible to determine the two extreme points for each attribute, and evaluate those to find the maximal possible contribution towards the cumulative upper bound *under this particular support constraint*. By iterating over *all* possible support values, and selecting the maximal upper bound, one calculates a true upper bound, that is, one does not underestimate the highest future value. 0 and $sup(p, \mathcal{D})$ do not have to be considered since the first corresponds to a pattern not covering any instance and the second one to a pattern having the exact same score as the current one.

Determining the values of the extreme points is straight-forward as well: as we wrote above, the extreme points for a convex function lie on the border of the (convex) domain on which it is defined. Even if the support constraint rules out the corners of these borders, it still holds that the extreme points have to lie on either the axes or the lines connecting $sp(p, \mathcal{D})$ and those axes. This means in turn that given a constrained support value x_{sup}, the extreme points take

the form

$$\langle \max\{x_i^+, x_{sup}\}, \max\{0, x_{sup} - x_i^+\}\rangle \text{ and}$$

$$\langle \max\{0, x_{sup} - x_i^-\}, \max\{x_i^-, x_{sup}\}\rangle.$$

The algorithm for upper bound calculation for several target patterns is shown as Algorithm 4.

Algorithm 4 Multi-target upper bound calculation for $\sigma_{cum}(sp_e(p))$

Given: number of targets d, extended stamp point $sp_e(p) = \langle x_1^+, x_1^-, \ldots, x_d^+, x_d^-\rangle$, correlation measure σ
Return: upper bound ub_{cum} on $\sigma_{cum}(sp_e(p))$

$ub_{cum} = -\infty$
for $1 \leq x_{sup} \leq (x_1^+ + x_1^- - 1)$ **do**
  $ub = 0$
  for $1 \leq i \leq d$ **do**
    $ub^+ = \sigma(\langle \max\{x_i^+, x_{sup}\}, \max\{0, x_{sup} - x_i^+\}\rangle)$
    $ub^- = \sigma(\langle \max\{0, x_{sup} - x_i^-\}, \max\{x_i^-, x_{sup}\}\rangle)$
    $ub = ub + \max\{ub^+, ub^-\}$
  end for
  if $ub > ub_{cum}$ **then**
    $ub_{cum} = ub$
  end if
end for
return ub_{cum}

Complete mining can be used to find unordered sets, since satisfaction of a constraint is a binary decision that does not establish an order on the patterns or pattern sets. By defining a total order on the unordered set, it can be turned into an ordered set. This is what the CBA system does: it first performs complete level-wise mining, using the anti-monotone minimum support constraint, and then orders the result set according to $\leq_{CBA}$ of Definition 1.3.1. We will explain in Section 2.3 how to turn an unordered into a tree set.

2.1.2 Top-k mining

The second mining task is not concerned with finding *all* solutions that satisfy a certain constraint but is an optimization task. Given a measure σ, the goal is to find the k highest scoring patterns with regard to this measure on $\mathcal{D}$ (note that this k can be, and often is, 1).

We again choose the level-wise algorithm to illustrate mining. Algorithm 5 performs unpruned top-k search (for instance because the measure used is not boundable). This is once again an algorithm that would exhaustively enumerate the entire pattern space, testing the top-k constraint against every pattern. If σ is not boundable but has a known maximum value max (such as 1.0 for

Algorithm 5 The general level-wise algorithm for top-k mining

Given: language $\mathcal{L}$, refinement operator ρ, measure σ, data base $\mathcal{D}$, result list size k

Return: set of highest scoring k solutions $s \in \mathcal{L}$ according to σ on $\mathcal{D}$: $Th(\mathcal{L}_p, \mathcal{D}, \arg_k \max \sigma(sp(s)))$ or $Th(2^{\mathcal{L}_p}, \mathcal{D}, \arg_k \max \sigma(sp(s)))$

$\mathbb{S} = \emptyset, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  for all $s \in \mathbb{C}$ **do**
    if $|\mathbb{S}| < k$ **then**
      $\mathbb{S} = \mathbb{S} \cup s$
    else if $\sigma(sp(s)) > \min_{s' \in \mathbb{S}} \sigma(sp(s'))$ **then**
      $\mathbb{S} = \mathbb{S} \setminus s' \cup s$
    end if
  end for
  $\mathbb{P} = \mathbb{C}$
end while
return $\mathbb{S}$

confidence), mining can stop as soon as $\min_{p' \in \mathbb{S}} \sigma(sp(p')) = max$. Ideally, we would prefer an algorithm that can prune during mining. If a function is *boundable* (Definition 1.1.18), it is possible to prune using upper bound information, leading to Algorithm 6.

Taking care to select the right refinement operator

In the context of this algorithm employing upper bound-based pruning, we will now give a longer explanation of our statement from the introduction that an optimal refinement operator can be counter-productive in top-k mining. Consider Figure 2.10, where the lattice from Figure 2.1 is transformed into a tree, corresponding to the use of an optimal refinement operator.

Nodes are annotated with the score and an upper bound on the value of σ future descendants of this node can achieve, in the form $(score, upperbound)$. We assume that the tree is pruned during mining. In top-k mining, pruning takes the form that upper bounds are tested against the k-best score encountered so far (during the search) and the tree pruned once the upper bound falls below this score. We assume $k = 3$.

- When using breadth-first search, the entire first level will be evaluated. Since no pattern scores have been encountered in this step, $\{d\}, \{e\},$ and $\{a\}$ are included in the solution set.

  Due to the fact that the upper bounds of all singleton patterns exceed the third-best score, all patterns consisting of two items are enumerated and $\{a, b\}, \{c, d\}, \{c, e\}$ replace the three solutions from step one.

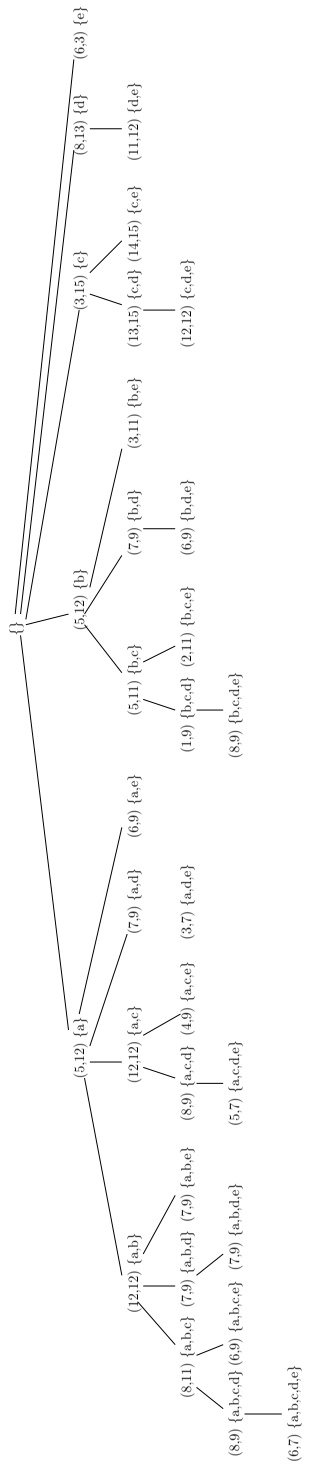


Figure 2.10: Itemset enumeration tree annotated with scores/upper bounds

Algorithm 6 The level-wise algorithm for top-k mining using pruning

Given: language $\mathcal{L}$, refinement operator ρ, boundable measure σ, data base $\mathcal{D}$, result list size k
Return: set of highest scoring k solutions $s \in \mathcal{L}$ according to σ on $\mathcal{D}$: $Th(\mathcal{L}_p, \mathcal{D}, \arg_k \max \sigma(sp(p)))$ or $Th(2^{\mathcal{L}_p}, \mathcal{D}, \arg_k \max \sigma(sp(s)))$

$\mathbb{S} = \emptyset, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  for all $s \in \mathbb{C}$ **do**
    if $|\mathbb{S}| < k$ **then**
      $\mathbb{S} = \mathbb{S} \cup s$
    else if $\sigma(sp(s)) > \min_{s' \in \mathbb{S}} \sigma(sp(s'))$ **then**
      $\mathbb{S} = \mathbb{S} \setminus s' \cup s$
    end if
  end for
  $\mathbb{P} = \{s \mid s \in \mathbb{C} \wedge ub(\sigma(sp(s))) > \min_{s' \in \mathbb{S}} \sigma(sp(s'))\}$
end while
return $\mathbb{S}$

The third-best score is 12, which means that only $\{c, d\}$ and $\{c, e\}$ are candidates for refinement. The optimal refinement operator only admits $\{c, d, e\}$, however, which is not added to the result set. In total, 16 potential solutions are enumerated.

- When using depth-first search, the enumeration is performed left-to-right, meaning that over-searching would happen. After the first level is enumerated and the singletons $\{d\}, \{e\}, \{a\}$ included in the solution set, only $\{a\}$ is refined.

  $\{a, b\}$ and $\{a, c\}$ are added and the third-best score increases to eight. In further steps, $\{a, b\}$ is refined but none of its descendants included, with the same holding for the rest of the "a"-branch and the entire "b"-branch. Once $\{c\}$ is refined, $\{c, d\}$ and $\{c, e\}$ are added to the solution set and the third-best score increases to 12.

  Finally, $\{c, d, e\}$ and $\{d, e\}$ are enumerated as well but not included in the solution set. In total, 21 potential solutions are enumerated.

- When using best-first search, the search which refines the most promising solutions first, that is, the ones with the largest upper bound, the first of the singleton solutions to be refined is be $\{c\}$.

  $\{c, d\}$ and $\{c, e\}$ are added to the solution set and the third-best score increases to 12. The next that is refined is $\{d\}$ since its upper bounds exceed 12 but the refinement is not added.

  Since we are aiming for general (short) solutions, $\{a\}$ is refined next,

and $\{a, b\}$ added to the solution set. 12 potential solutions have been enumerated.

In the case of depth-first search, the ostensibly more efficient optimal refinement operator, due to the avoidance of duplicate enumeration, actually leads to a larger part of the search space being traversed then in breadth-first search. In the case of best-first search based on the upper bound, while it is the most efficient solution in this case, the disregard of areas of the search space can lead to a slow increase of the upper bound and thus expensive mining operations. To summarize, in breadth-first mining, an optimal refinement operator is usually effective, since it avoids the duplicate enumeration that otherwise occurs. In depth-first and best-first search, different refinement operators are often more suited.

A final remark to which we will return later: Top-k mining establishes an inherent order on the patterns mined: the ranking according to the value of $\sigma(sp(p))$. This does not mean that this is the only possible order on the result set or even that the result set does have to be treated as an ordered set.

2.1.3 Exhaustive post-processing for pattern set mining

Both local pattern mining and pattern set mining can be performed exhaustively, either in a complete or top-k manner. This leads to an approach consisting of two phases, as shown in Algorithm 7.

Algorithm 7 Two-phase exhaustive search for pattern set mining

Given: pattern language $\mathcal{L}_p$, data base $\mathcal{D}$, local pattern constraint c, pattern set constraint $\mathcal{C}$
Return: $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

Exhaustively mine $\mathbf{L} = Th(\mathcal{L}_p, \mathcal{D}, c)$
Exhaustively mine $\mathbf{M} = Th(\mathbf{L}, \mathcal{D}, \mathcal{C})$
return $\mathbf{M}$

There have been works that performed these kinds of mining operations, most notably those by Shima *et al.* (2005) and Knobbe *et al.* (2006a, 2006b).

Shima *et al.* define a redundancy constraint that does not consider patterns in a pairwise manner (see Equation (1.3.24)). To deal with the problem of a too large search space, they limit the patterns used to the maximal frequent patterns, and a depth-first search strategy. They interpret the resulting pattern sets as DNF formulae with high coverage and small overlap between the clauses but do not generalize their setting to more complex constraints or settings. As our reformulation in Equation (1.3.24) showed, their setting is a special case of constraint-based pattern set mining.

Knobbe *et al.* aim to assemble sets of patterns that satisfy various, sometimes mutually exclusive, constraints, chiefly one of small pairwise redundancy. Their solution to navigating the large search space lies in explicitly limiting the

cardinality of pattern sets (or *pattern teams*, as they label them). In (Knobbe and Ho 2006a), redundancy reduction is achieved by maximizing joint entropy in a level-wise manner, using upper-bound pruning, effectively enforcing the constraint:

Definition 2.1.5. Maximally informative k-itemsets constraint

$$\mathcal{C}_{miki}(\mathbb{S}, \mathcal{D}) \equiv \arg_1 \max j\text{-}ent(\mathbb{S}, \mathcal{D}) \wedge size(\mathbb{S}) = k$$

Originally conceived as an itemset mining technique, that is, local pattern mining, the authors show that it can be used for pattern set mining. Despite these parallels, Knobbe *et al.* also focus on a rather restricted setting (which includes the mandatory size constraint) and do not generalize towards a notion of constraint-based pattern set mining.

In contrast to these approaches, we introduced a general technique, to the best of our knowledge the first such system, in (De Raedt and Zimmermann 2007). Our formulation allows the use of arbitrary constraints and uses pruning based on monotonicity and boundable constraints. We will evaluate it experimentally in Section 3.1.

As we stated at the beginning of this section, exhaustive algorithms have two advantages:

1. Only exhaustive algorithms are *complete*, that is return all patterns that satisfy a given constraint

2. Exhaustive algorithms *guarantee* that the best solutions are found in an optimization setting

A complete mining result is not always necessary, though. Especially in the setting of pattern set mining, often a single pattern set is all that is desired by the end user for the next processing step. Additionally, given the weaker pruning power of boundable constraints when compared to anti-monotone constraints, the combinatorial explosion originating in a large number of local patterns becomes hard to control. Therefore, optimization mining tends to use significant running times and resources. The alternative lies in the use of heuristic methods, which are faster but sacrifice guarantees about the quality of found solutions.

2.2 Heuristic Techniques for Mining

The goal of heuristic search is to limit the size of the search space that has to be traversed. While exhaustive search uses pruning to achieve this goal, pruning techniques will not necessarily lead to an effective decrease in size. The heuristic approach consists of choosing one (or several) potential solutions, refining those into "neighboring" solutions and selecting one (or several) of these for further refinement.

Due to the fact that the non-completeness is inherent in heuristic approaches, it is impossible to perform complete mining using such techniques. The following

Algorithm 8 The general hill-climbing algorithm

Given: an initial solution $s_{in} \in \mathcal{L}$, optimization function ϕ
Return: a locally optimal solution s_{opt}

$s_{old} = \varnothing$
$s_{opt} = s_{in}$
while $s_{opt} \neq s_{old}$ **do**
  $s_{old} = s_{opt}$
  $\mathbb{C} = \{s_{cand} \mid s_{cand} \in \rho(s_{opt})\}$
  $\mathbb{P} = \{s \mid s \in \mathbb{C} : \phi(s) > \phi(s_{opt})\}$
  select one $s \in \mathbb{P}$: $s_{opt} = s$
end while
return s_{opt}

explanations will therefore focus on the optimization setting. It helps if we keep the itemset lattice in mind for this purpose.

An often employed class of heuristic algorithms performs so-called *hill-climbing*. The general hill-climbing algorithm is given as Algorithm 8. A hill-climbing algorithm iteratively makes small changes to a given solution, each time improving it a bit. It terminates when there is no further improvement possible anymore. This will lead to the discovery of *locally* optimal solutions, without any guarantee of the solution being globally optimal, however. An additional aspect that has to be considered is the selection of the next solution to be refined: this selection can happen arbitrarily, randomly or by selecting $\arg\max_{s \in \mathbb{P}} \phi(s)$, for instance. If the last selection criterion is chosen, the heuristic algorithm is called *greedy* (Russell and Norvig 2003):

Property. *A* greedy *algorithm is a heuristic algorithm that at each stage makes the* locally optimal *choice with the aim of finding a globally optimal solution.*

Since exhaustive mining can be rather expensive and for certain settings, such as when patterns are used as features for classification, a single pattern set is all that is needed, we developed a greedy algorithm, called PICKER, in (Bringmann and Zimmermann 2009). We present experimental results regarding it in Chapter 3, Section 3.2.

2.2.1 Beam search

To overcome the limitations of hill-climbing search without incurring the high computational costs of exhaustive mining, beam search is an often used technique in heuristic local pattern mining. Beam search is effectively *parallel* hill-climbing, that is, several solutions are improved at the same time (Mitchell 1997). The main improvement lies in that *several* solutions are kept for future refinement, hopefully avoiding local maxima.

Considering the general formulation of a beam search algorithm (Algorithm 9), one sees that an additional parameter has to be specified, the beam size

Algorithm 9 The general beam search algorithm

Given: language $\mathcal{L}$, refinement operator ρ, measure ϕ, data base $\mathcal{D}$, beam size b
Return: a locally optimal solution s_{opt}

$s_{opt} = \varnothing, \mathbb{P} = \{\epsilon\}$
while $\mathbb{P} \neq \emptyset$ **do**
  $\mathbb{C} = \{s \mid \exists s' \in \mathbb{P} : s \in \rho(s')\}$
  if $\max_{s \in \mathbb{C}} \phi(s) > \phi(s_{opt})$ **then**
    $s_{opt} = \arg\max_{s \in \mathbb{C}} \phi(s)$
  end if
  $\mathbb{P} = \{s \mid s \in \mathbb{C}, s$ among the b highest-scoring solutions $\in \mathbb{C}$ acc. to $\phi\}$
end while
return s_{opt}

b. Smaller beam sizes speed up the mining process at the cost of a higher probability to arrive at a local optimum that is not the global one. Larger beam sizes on the other hand make the mining behavior approach that of an exhaustive algorithm, with the computational costs attached. The stopping criterion we give here depends on a finite language but other criteria could be applied for inclusion of solutions in $\mathbb{P}$. Local pattern mining algorithms such as CN2 (Clark and Niblett 1989), or RIPPER (Cohen 1995) use non-optimal refinement operators to ensure that all neighboring solutions can be reached from a given candidate solution.

Top-k optimization mining is in general possible using beam-search, for instance by setting the beam size to k. But there is no guarantee that different refinement-paths do not lead to the same solution or that the result set really is an approximation of the k best patterns.

2.2.2 Order-restricted hill-climbing

Hill-climbing and beam-search (especially for small beam sizes) reduce the computational cost of mining by considering only a few potential solutions for expansion in each iteration, restricting the search space. Depending on the refinement operator, if is ideal, for instance, such an approach can be problematic, if too many potential refinements exist.

Consider as an example pattern set mining as a post-processing step of a local pattern mining operation. Since the result set can easily consist of several thousands or tens of thousands of patterns, even a beam size of 1 will be problematic, if *all* refinements are created and no efficient pruning method exists. For a result set of 1000 *patterns*, the potential solution s_{opt} has to be extended by $1000 - |s_{opt}|$ in *each iteration* and each of these refinements evaluated.

It is therefore fitting that it was in this context (Liu et al. 1998), that a different kind of search was pioneered, that we will refer to as *order-restricted*

hill-climbing. The general idea is that an order is defined on the elements which can be added to a solution, only those can be added that are *greater* than the ones already present, according to the order. This can be translated into an ordered refinement operator.

Definition 2.2.1. Ordered Refinement Operator *Given a pattern language $\mathcal{L}_p$ and an order $\trianglelefteq$ defined on it, an* ordered *refinement operator takes the form:*

$$\rho_{\trianglelefteq}(\mathbb{S}) = \{\mathbb{S} \cup \{p\} \mid p \in 2^{\mathcal{L}_p}, p \trianglerighteq \max_{\trianglelefteq}\{p' \mid p' \in \mathbb{S}\}\}$$

It can be seen that this is an optimal refinement operator in that each solution can be reached by exactly one path through the search tree, giving rise to an enumeration tree like the one in Figure 2.10. The data base coverage pruning scheme used in CBA (Section 4.2), and our BOUNCER algorithm (Section 3.2) are instantiations of order-restricted heuristic pattern set mining.

In the unlikely worst case, nothing is gained compared to the unrestricted case, since it could happen that each pattern is added to the set and therefore in each iteration $1000 - |s_{opt}|$ refinements are enumerated.

As soon as a pattern is encountered that improves $\mathbb{S}$, the evaluation of refinements is stopped and this pattern added to the set in CBA and BOUNCER. This can be formalized as selection operator of the form:

$$s_{opt} = \{\mathbb{S} \cup \{p\}\} \in \mathbb{P} \text{ such that } p \text{ minimal under } \trianglelefteq$$

While this is rather efficient, it severely limits the optimality of the approach. By choosing the "first" good solution, the risk of arriving at a local optimum is rather high.

We show the general order-restricted hill-climbing algorithm as Algorithm 10, and will further discuss the used orders and the choice of ϕ in the sections describing the application of those algorithms to pattern set mining.

Algorithm 10 The general order-restricted pattern set mining algorithm

Given: pattern language $\mathcal{L}_p$, order $\trianglelefteq$, optimization function ϕ
Return: a locally optimal solution $\mathbb{S}_{opt}$

$\mathbb{S}_{old} = \emptyset$
$\mathbb{S}_{opt} = \{\epsilon\}$
while $\mathbb{S}_{opt} \neq \mathbb{S}_{old}$ **do**
  $\mathbb{S}_{old} = \mathbb{S}_{opt}$
  $\mathbb{C} = \{\mathbb{S}_{cand} \mid \mathbb{S}_{cand} \in \rho_{\trianglelefteq}(\mathbb{S}_{opt})\}$
  $\mathbb{P} = \{\mathbb{S} \mid \mathbb{S} \in \mathbb{C} : \phi(\mathbb{S}) > \phi(\mathbb{S}_{opt})\}$
  Select $s_{opt} = \{\mathbb{S} \cup \{p\}\} \in \mathbb{P}$ such that p minimal under $\trianglelefteq$
end while
return $\mathbb{S}_{opt}$

2.2.3 Upper-bound ordered hill-climbing

We relativized our statement regarding the potential inefficiency of hill-climbing above by "if no efficient pruning method exists". If the improvement gained by adding an element to a solution is upper-boundable, it is possible to sort elements by decreasing upper bound. In each iteration refinements are then enumerated and evaluated one after another and enumeration stopped once a solution has been found that has a value that exceeds the upper bounds of the remaining elements. The advantage is that this algorithm can select s_{opt} in a greedy manner, unlike order-restricted approaches.

Definition 2.2.2. Upper-bound Ordered Refinement Operator *Given a pattern language $\mathcal{L}_p$, a quality measure Φ and an order $\trianglelefteq_{ub} : \forall p_1, p_2 \in \mathcal{L}_p, p_1 \triangleleft_{ub} p_2 \Leftrightarrow ub_\Phi(p_1) > ub_\Phi(p_2)$ defined on it, an* upper-bound ordered *refinement operator takes the form:*

$$\rho_{\trianglelefteq_{ub}}(\mathbb{S}) = \{\mathbb{S} \cup \{p\} \mid p \in \mathcal{L}_p, \neg\exists p' \in \mathcal{L}_p, p' \triangleleft_{ub} p : \Phi(\mathbb{S} \cup \{p'\}) \geq ub_\Phi(p)\}$$

One aspect of this refinement that is immediately obvious is that each candidate solution has to be evaluated immediately after it is enumerated, to give the refinement operator the information needed for pruning. Less obvious is that contrary to order-restricted search the order used in the refinement operator can be dynamic, if upper bounds are recalculated during the search process. By using upper-bound ordered hill-climbing, we upgrade the BOUNCER algorithm, to PICKER* (Section 3.2), returning a better solution. The experimental evaluation and comparison of these techniques can be found in Section 3.2.

2.2.4 Heuristic post-processing techniques for pattern set mining

Generally, as in the exhaustive case, heuristic algorithms can be used to mine both local patterns and pattern sets, reducing computational complexity. Since all the techniques shown here assume a rather large alphabet with which to enumerate potential solutions, heuristic search cannot typically be used as the first phase of pattern set mining, however. A general heuristic pattern set mining approach would thus look like Algorithm 11.

Algorithm 11 Two-phase exhaustive-heuristic search for pattern set mining

Given: pattern language $\mathcal{L}_p$, data base $\mathcal{D}$, local pattern constraint c, pattern set constraint $\mathcal{C}$
Return: $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

Exhaustively mine $\mathbf{L} = Th(\mathcal{L}_p, \mathcal{D}, c)$
Heuristically mine $\mathbf{M} = Th(\mathbf{L}, \mathcal{D}, \mathcal{C})$
return M

This is the basic template for pattern set mining that has been introduced in the data mining literature so far. CBA and CMAR use heuristic post-processing steps to assemble sets of predictive rules. Both techniques use order-restricted hill-climbing, based on $\trianglelefteq_{CBA}$, introduced in Example 1.3.1, and CBA imposes the constraint introduced as Example 1.3.4. CMAR relaxes this constraint by allowing each instance to be covered by several patterns, asking the user to specify how many, leading to the constraint:

Definition 2.2.3. CMAR constraint

$$\mathcal{C}_{CMAR}(\mathbb{S}, \mathcal{D}) \equiv \forall p \Rightarrow q \in \mathbb{S}, \exists e \in \mathcal{D} : |\bigcup_{p_i \trianglelefteq_{CBA} p} cov(p_i, e)| < \theta_{db-cov} \wedge class(e) = q$$

Siebes *et al.* (2006) use the MDL principle to select from a set of frequent patterns mined in a local pattern mining operation those that have low redundancy and describe the underlying data accurately by compressing it. The authors mine an ordered pattern set that is used as a code table to encode the database. The order on this pattern set takes the form:

Definition 2.2.4. Compression Set Order *Given two itemsets p_1, p_2, we say that $p_1 \trianglelefteq_{coding} p_2$ if and only if:*

- $|p_1| > |p_2|$ *or*
- $|p_1| = |p_2| \wedge sup(p_1) > sup(p_2)$

Starting from the set of all *singleton* items, candidate patterns are considered in the order:

Definition 2.2.5. KRIMP **mining order** *Given two itemsets p_1, p_2, we say that $p_1 \trianglelefteq_{Krimp} p_2$ if and only if:*

- $sup(p_1) > sup(p_2)$ *or*
- $sup(p_1) = sup(p_2) \wedge |p_1| > |p_2|$

as made explicit in (van Leeuwen et al. 2006). It should be noted that this order is reversed from the one upheld on the pattern set itself. For each candidate pattern p_{cand}, it is checked whether the encoding using $\mathbb{S} \cup \{p_{cand}\}$ is smaller than the one using $\mathbb{S}$. If so, the pattern is accepted, otherwise accepted. Noting that this fixed order can lead to suboptimal pattern sets, they develop their technique towards one that checks in each iteration whether removing an already included pattern can lower the size of the encoding. This kind of heuristic search, akin to neighborhood search, therefore increases the opportunities for finding an optimal pattern set.

Reducing a set of patterns to a subset that captures the semantics of the original set while being compact, is a task that has been addressed in many different ways over the years. The importance of selecting appropriate features for machine learning tasks to keep concept learners from being overwhelmed was addressed ten years ago in (Liu and Motoda 1998). Guyon *et al.* (2003)

gives a more recent overview of motivations and techniques for feature selection. According to the authors, the approaches can be distinguished into *embedded, wrapper*, and *filter* techniques.

This connection is pointed out by Knobbe *et al.* (2006b). In addition to the solutions towards finding pattern teams described in Section 2.1.3, the authors also present heuristic methods optimizing their quality criteria. The technique proposed in (Cheng et al. 2007) greedily selects the pattern maximizing a gain-function. This means that no mining order is used but instead *all* remaining patterns have to be checked. A pattern added to the set if it satisfies the same constraint that CMAR uses.

In (Bringmann and Zimmermann 2009), we proposed an order-restricted hill-climbing algorithm for mining pattern sets called BOUNCER, using the second constraint defined in Example 1.3.4. The main difference to existing techniques lies in the fact that the order used is not fix but can be decided by the user. We also proposed a greedy technique using upper-bound ordered hill-climbing to abstract from the effects of orders used (PICKER*). We evaluate both techniques in Section 3.2.

The ORIGAMI algorithm proposed in (Hasan et al. 2007) deviates from the other techniques discussed here in that the local pattern mining step is heuristic. The authors acknowledge the problems inherent in finding a representative set of local patterns by sampling but point towards the high complexity of their domain to motivate their choice. They propose two heuristic techniques. In the first, patterns that have a maximum similarity to patterns already included in the set (governed by a similarity threshold) are selected at random and added to the set. The constraint therefore takes the form:

Definition 2.2.6. ORIGAMI constraint

$$\mathcal{C}_{ORIGAMI}(\mathbb{S}) \equiv \forall p_i, p_j \in \mathbb{S}, p_i \neq p_j : sim(p_i, p_j) \leq \theta_{sim}$$

This approach is *not* greedy since the authors do not adopt a notion of *best* pattern with regard to the pattern set in selecting with which pattern to extend the pattern set. In the second algorithm, pattern set mining starts from a random pattern set and performs hill-climbing through the space of neighboring solutions, replacing a pattern satisfying the constraint by another one, until no improvement is possible.

All of those approaches trade-off the computational complexity against the quality of solutions. Depending on the context in which mining is used, patterns that are not globally optimal can still be combined into an effective set, however. As we will see in Chapter 5, there are iterative mining techniques that correct suboptimal patterns when mining a complete set. In this way, iterative approaches, which we will describe in the next section, are more powerful than the post-processing methods of the present and the preceding section.

2.3 Iterative Mining

The exhaustive and heuristic techniques described in the preceding two sections were formulated in such a way that potential solutions are evaluated on the whole data during the search. This is well-suited for the case of explicitly defined constraints that capture *exactly* what the user wants. An alternative lies in an implicit formulation that generally takes the form:

Mine patterns that are interesting when considering the patterns that have so far been mined.

A straight-forward way of doing this consists in manipulating the data according to patterns' effects before mining further patterns. General speaking, iterative mining consists of:

1. Mining $Th(\mathcal{L}_p, \mathcal{D}, c)$
2. Evaluate a pattern set constraint $\mathcal{C}$ and if satisfied, add $Th(\mathcal{L}_p, \mathcal{D}, c)$ to the pattern set, otherwise stop the mining process
3. Manipulate the data, depending on $Th(\mathcal{L}_p, \mathcal{D}, c)$
4. Return to 1.

This is the approach that has been taken to pattern set mining in most of the machine learning literature so far. In the case of concept learning, two early paradigms are *sequential covering* (Clark and Niblett 1989) or *separate and conquer*, and *parallel covering* or *divide and conquer* (Quinlan 1993). In the former technique, instances that are covered by $Th(\mathcal{L}_p, \mathcal{D}, c)$ are removed before the next iteration, leading to decision lists (ordered sets of predictive rules). In the latter, data are divided into covered and uncovered data and mining re-iterated on the subsets, leading to decision trees (tree sets).

In both cases, the local pattern mining step (beam search for sequential covering and selecting the most discriminative attribute-test in parallel covering) has been considered largely coupled to the pattern set mining step itself. We abstract from this view and discuss the two paradigms in the more general terms *sequential mining* and *parallel mining* in this section. This allows us to incorporate other choices for the local pattern mining and data manipulation steps.

2.3.1 Sequential mining

The sequential mining approach takes the form shown as Algorithm 12. Initially, all instances are given the same consideration for local pattern mining. Depending on the patterns mined and their interpretation, for instance, descriptive or predictive, the weights of individual instances are then adjusted. This means that patterns are ordered by when they have been mined, with all patterns:

Definition 2.3.1. Sequential Set Order *Given a pattern set* $(\mathbb{S}, \trianglelefteq_{seq})$ *mined by sequential mining,* $\forall p \in \mathbb{S} : p' \trianglelefteq_{seq} p \Leftrightarrow p'$ *was mined in an earlier iteration than* p

Algorithm 12 The general sequential pattern set mining algorithm

Given: pattern language $\mathcal{L}_p$, data base $\mathcal{D}$, local pattern constraint c, pattern set constraint $\mathcal{C}$
Return: $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

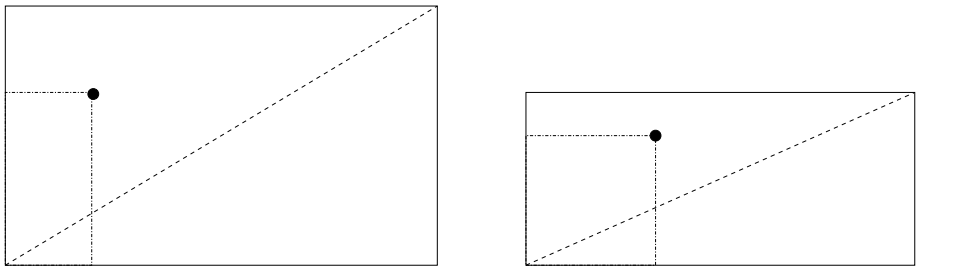
$\mathbb{S} = \emptyset$, $\forall e \in \mathcal{D} : weight(e) = 1$
while the stopping criterion is not satisfied **do**
  Mine $Th(\mathcal{L}_p, \mathcal{D}, c)$
  if $Th(\mathcal{L}_p, \mathcal{D}, c) = \emptyset$ **then**
    return $\mathbb{S}$
  else if $\mathcal{C}(\mathbb{S} \cup Th(\mathcal{L}_p, \mathcal{D}, c)) = true$ **then**
    $\mathbb{S} = \mathbb{S} \cup Th(\mathcal{L}_p, \mathcal{D}, c)$
  else
    return $\mathbb{S}$
  end if
  $\forall e \in \mathcal{D}$ adjust $weight(e)$
end while
return $\mathbb{S}$

In this work, we will focus on two main solutions for adjusting the weights of instances in the data, known in machine learning as *weighted covering* and *sequential covering*:

1. Re-weighting of covered (matched) instances: the weight of an instance is decreased to a value greater than 0, if it is covered by $Th(\mathcal{L}_p, \mathcal{D}, c)$
2. Removal of covered (matched) instances: all instances covered by $Th(\mathcal{L}_p, \mathcal{D}, c)$ are removed before new patterns are mined, which corresponds to setting their weight to 0.

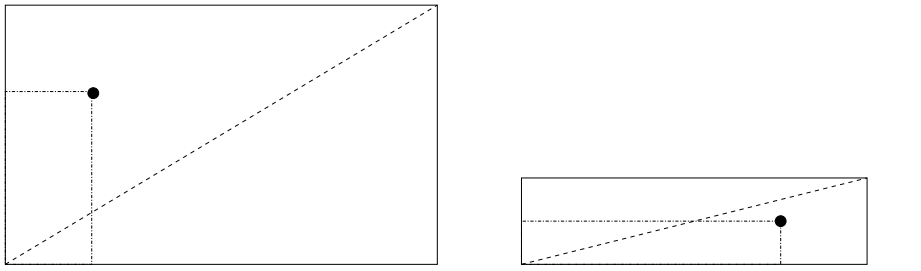
$Th(\mathcal{L}_p, \mathcal{D}, c)$ can be a single rule, as in CN2 or CN2-SD (Lavrač et al. 2004), but also a set of rules, as in the ART system (Galiano et al. 2004), or even more complex patterns. There is an implicit assumption in both of these techniques, namely, that phenomena can be described by a single theory $Th(\mathcal{L}_p, \mathcal{D}, c)$. Once instances have been covered (matched) by a theory, they are diminished or removed from future consideration, making $Th(\mathcal{L}_p, \mathcal{D}, c)$ their prime description.

To visualize how the distribution against which interestingness measures are evaluated changes by re-weighting during mining, consider Figure 2.11. For the sake of simplicity we assume the constraint $\arg_1 \max \sigma(sp(p))$ during the local pattern mining step. On the left-hand side the original distribution of the target attribute in the data is shown, as well as the highest-scoring pattern according to σ. By re-weighting covered instances, the coverage space contracts, which means that covered instances have less influence on the score of future patterns. Sequential re-weighting will direct the local pattern mining to uncovered subsets of the data, thus adding patterns to $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$ that are different from the ones already included.



(a) Starting distribution (b) Distribution after the first re-weighting

Figure 2.11: Changes in target attribute's distribution during re-weighting



(a) Starting distribution (b) Distribution after the first removal

Figure 2.12: Changes in target attribute's distribution during removing

This process can be slow, however, depending on the aggressiveness of the re-weighting scheme. Also, if the scheme is not aggressive enough, it can lead to patterns that while different still show high similarity in performance or structure. If the covered data is removed, it is *guaranteed* that new patterns are mined on uncovered data. This will speed up the pattern set mining process, and ensure that the patterns included are relatively dissimilar. Consider Figure 2.12 to see this effect. The coverage space shrinks repeatedly, which means both that fewer patterns will be truly interesting and that interesting patterns will show very different behavior from each other. On the other hand, the influence of already covered data is completely removed from consideration, which can be unrealistic with local phenomena that are not necessarily disjunct.

2.3.2 Sequential pattern set mining

Sequential covering has been discussed in the machine learning literature for quite a long time already, especially in the context of learning rules for classification purposes. The primary example is of course Clark and Niblett's CN2 (Clark and Niblett 1989). Adhering to the same principles is the FOIL system (Quinlan 1990), which upgraded sequential covering to first-order logic representations of rules. The culmination is arguably the RIPPER system (Cohen 1995). Similarly to the way Siebes *et al.*'s MLD-approach tries to overcome local optima by revisiting made choices, RIPPER performs neighborhood search to improve a found pattern set. Instead of possibly deleting patterns, the attempt is made to replace them either by a specialization or by a newly enumerated pattern.

A relatively recent system that uses sequential mining for classification purposes is ART (Galiano et al. 2004). In each iteration a variable number of minimum-support high-confidence class-association rules is mined, the training examples covered by the selected rules removed and the mining step reiterated. Due to the fact that the technique ensures that only mutually exclusive rules are selected in each iteration, there is no order between those rules, leading to a set that is not exactly an ordered set in the sense of Definition 1.3.2.

A very recent system in which the mining process does not have to be restarted every time is the DDPMINE algorithm proposed in (Cheng et al. 2008). Using the FP-GROWTH (Han et al. 2000) technique, an *FP-Tree* is constructed over a database which adheres to some minimum constraints. The best class-correlating pattern is selected and instances that are covered by it removed. The updated FP-Tree is then used to select the next-best feature and this process reiterated. The authors point out the connection to sequential covering themselves in their work and their approach has the advantage of essentially using only one mining operation. They run the risk of exhausting working memory, however, since the constraint on the FP-Tree needs to be lenient enough not to rule out interesting features in future iterations.

The idea of sequential re-weigthing was first introduced by Gamberger *et al.* (2000), using a similar argumentation to the one we sketched at the beginning of this section. The authors expanded on their work, amongst others in (Gamberger and Lavrac 2002) and of course the work in which CN2-SD was

developed, (Lavrač et al. 2004). This last work is especially noteworthy since its comparison to CN2 in the context of concept learning showed similarly good performance at reduced rule set size. This gives additional support to the argument that completely removing covered instances places a too heavy burden on the local pattern mining step.

Another work that used a re-weighting approach is (Yin and Han 2003), which proposed the CPAR system. The re-weighting idea in this work arises from the attempt at covering each instance multiple times, mirroring their approach in the CMAR system, developed by the same group.

Analyzing CN2 and CN2-SD led us to replace the heuristic local pattern mining steps by exhaustive techniques and we evaluate the resulting systems, CN2_{CG} and CG-SD, in Sections 5.1 and 5.2, respectively.

Related to yet different from re-weighting is the machine learning technique known as *boosting*, which was introduced in the context of classification. The general approach to boosting takes the form:

1. Mine a predictive pattern.
2. Increase the weight of misclassified instances.
3. Return to 1)

Boosting is different from re-weighting as we have described it here in that instances that are uncovered by any of the patterns but influenced by their interpretation (via classification) have their weight adjusted. In the case of "true" boosting techniques, such as AdaBoost (Freund and Shapire 1996) or LPBoost (Demiriz et al. 2002), the weight-adjustment can be proven to converge the classifier to 100% accuracy. The boosting scheme has, as other machine learning techniques, not found wide-spread application in the data mining community. One notable exception is the work by Kudo *et al.* (2004). The authors show that their algorithm improves on methods using baseline (constraint-satisfaction) features, supplying more support for the idea that sequential mining is a powerful way of iterative pattern mining.

2.3.3 Parallel mining

Re-weighted of data alters the focus of the mining process slowly, assembling a set of patterns that describe local phenomena which, while different, might still overlap. Removal of data achieves this shift of focus far more quickly, ensuring that patterns are more different. As we mentioned before, there is an implicit assumption in both of these techniques, namely that phenomena can be described by a single theory. While other theories might overlap (at least in the case of re-weighting) with this one, the phenomena they describe are supposed to be different ones.

There is another approach to assembling a set of patterns, however, which is directly tied to the notion of tree sets. Inspired by decision trees, in this approach a theory is not generally assumed to describe a phenomenon satisfactorily. Instead, the covered data is mined again in case that there are more

fine-grained phenomena which can be described in more detail. The uncovered data is of course mined as well. The manipulation of the data therefore takes the form of a *split* into covered and uncovered parts.

Algorithm 13 The general parallel pattern set mining algorithm

Parallel-Mine
Given: pattern language $\mathcal{L}_p$, data base $\mathcal{D}$, local pattern constraint c, pattern set constraint $\mathcal{C}$
Return: $Th(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

$\mathbb{S} = Th(\mathcal{L}_p, \mathcal{D}, c)$
if $\mathbb{S} = \emptyset$ **then**
  return $\mathbb{S}$
end if
$\mathcal{D}_{cov} = cov(Th(\mathcal{L}_p, \mathcal{D}, c), \mathcal{D})$
$\mathcal{D}_{uncov} = \overline{cov}(Th(\mathcal{L}_p, \mathcal{D}, c), \mathcal{D})$
if $\mathcal{C}(\mathbb{S} \cup$ *Parallel-Mine*$(\mathcal{L}_p, \mathcal{D}_{cov}, c, \mathcal{C})) = true$ **then**
  $\mathbb{S} = \mathbb{S} \cup$ *Parallel-Mine*$(\mathcal{L}_p, \mathcal{D}_{cov}, c, \mathcal{C})$
end if
if $\mathcal{C}(\mathbb{S} \cup$ *Parallel-Mine*$(\mathcal{L}_p, \mathcal{D}_{uncov}, c, \mathcal{C})) = true$ **then**
  $\mathbb{S} = \mathbb{S} \cup$ *Parallel-Mine*$(\mathcal{L}_p, \mathcal{D}_{uncov}, c, \mathcal{C})$
end if
return $\mathbb{S}$

The general parallel mining algorithm is given as Algorithm 13. The resulting pattern set is a tree pattern set: each pattern is element of a total order with regard to

- The patterns that gave rise to the subset on which it was mined, those come before it in the order.
- The patterns that are mined on the subsets it gave rise to, those come after it.

The algorithm works recursively in that it calls itself on the subsets of the data which are covered and uncovered by $Th(\mathcal{L}_p, \mathcal{D}, c)$, respectively. In the case that such a local pattern theory cannot be mined, it is impossible to split the data into subsets, leading to the termination of the particular recursive call. Additionally to the local pattern constraint c, there is also a pattern set constraint $\mathcal{C}$ involved, which might preclude the result of a local pattern mining operation to be added to the pattern set. The pattern set constraint has to be checked twice, for $Th(\mathcal{L}_p, \mathcal{D}, c)$ mined on the covered data $\mathcal{D}_{cov}$ and the uncovered data $\mathcal{D}_{uncov}$, respectively.

It is of course possible that the found theories at each stage describe the data perfectly, in which case re-mining on the covered subset will not yield more patterns. A tree set can therefore degenerate into an ordered set as we

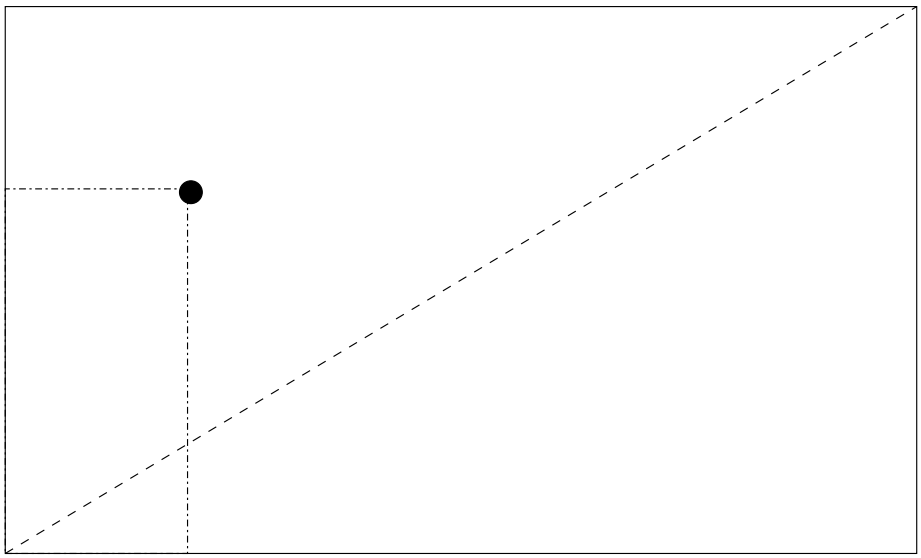
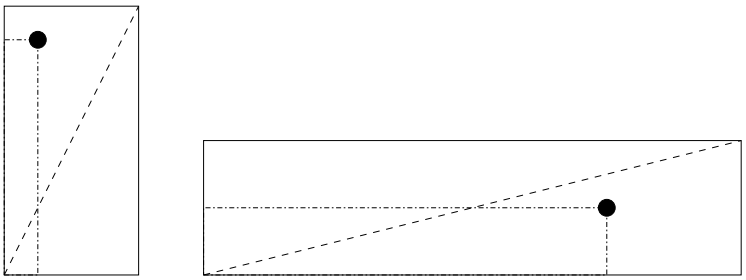


Figure 2.13: Original distribution

explained before. To contrast the manipulation of the data against the choices made in sequential mining, consider Figures 2.13 and 2.14.

The starting coverage space is the same as in the illustration of sequential mining but mining of a theory has the effect of multiplying coverage spaces. In sequential mining only one such space has to be considered at each stage. The right-hand side of Figure 2.14 shows the same distribution as exists after the first iteration of the sequential removal. The left-hand side on the other hand is completely new, not discounted data that remains in the coverage space as in re-weighting. If there is a phenomenon that can be described in more detail by an additional theory, this theory will be found, contrary to removal or re-weighting of the data.



(a) Covered data (b) Uncovered data

Figure 2.14: Changes in target attribute's distribution after the first split

By discarding the strong assumption of sequential mining and keeping data for re-mining, tree pattern sets become more expressive. This can be seen by the fact that tree sets can turn into ordered sets as well. This is accompanied by the usual trade-off however: keeping the covered data and re-mining it can be expected to lead to more computational effort than will be needed for the sequential technique.

2.3.4 Parallel pattern set mining

The parallel mining technique originates in decision tree learners, of which C4.5 (Quinlan 1993) is the best-known system. Decision trees can be interpreted in different ways:

- As tree sets of low-complexity patterns. In each iteration, a single attribute-value pair is chosen to split the data, corresponding to elements of a subset of $\mathcal{L}_{AV}$.
- As an unordered sets of conjunctive patterns since each branch can be interpreted as a conjunctive pattern and all branches are mutually exclusive.
- As a local pattern, for instance when decision trees are *boosted*.

In terms of the first interpretation, there exists work in the machine learning literature that aims at increasing the complexity of individual patterns. Combining tests linearly has been proposed in (Murthy 1997), and Blockeel *et al.* have proposed using conjunctions of first-order predicates for clustering (Blockeel et al. 1998). Using exhaustive techniques, we instead propose the mining of conjunctive rules for conceptual cluster formation by the CG-Clus system, evaluated in Section 6.1. Such a technique also allows us to efficiently mine *sets* of patterns in *each* iteration, leading to tree sets that we call *Ensemble-Trees*. We study their application to classification in Section 6.3.

The parallel mining paradigm has not seen much use in the data mining community so far. An exception is DT-GBI technique of Geamsakul *et al.* (2003). Staying true to the machine learning origins, the authors use beam-search to mine a discriminative graph patterns in each iteration. More recently, Fan *et al.* adopted parallel mining for deriving patterns used as features in a classification task (Fan et al. 2008). Independently from those two approaches, we proposed parallel mining for decision tree induction on tree-structured data (Bringmann and Zimmermann 2005). The resulting algorithm, Tree^2, is experimentally evaluated in Section 6.2.

2.3.5 Iterative pattern set mining beyond sequential and parallel mining

In general, iterative mining does not have to proceed in a sequential or parallel way. An example of an iterative mining technique that is performed differently was proposed by Rückert *et al.* (2007). Using a stochastic random walk,

the authors propose mining class-correlating graph patterns, and iterating this process, maximizing a measure trading off class-correlation and similarity to patterns mined before. Those patterns are then shown experimentally to be more suitable for classification tasks, leading to smaller feature sets, better performance or both.

In a slightly different vein are the nFoil and kFoil approaches by Landwehr *et al.* (2005, 2006). The goal of these works is the induction of clauses in first-order logic. The authors consider the task to be *dynamic propositionalization* and induce the clauses iteratively. The quality of new candidates is evaluated by the performance of the candidate pattern (that is clausal) sets in a classifier (Naïve Bayes and an SVM, respectively). In contrast, in our formulation of the iterative pattern set mining task, patterns are (explicitly or implicitly) evaluated in terms of the instances they cover, not by their usefulness. The main ideas of both approaches are the same, however.

2.4 Summary

After laying out the basics of pattern and pattern set enumeration and the properties and constraints to mine them, we used the third chapter to describe various algorithmic solutions to actually deriving the patterns and sets of patterns we aim for. We split the chapter into three section, devoted to exhaustive, heuristic and iterative mining, respectively.

Exhaustive search is the most reliable in a certain sense since optimal solutions can be guaranteed and that complete mining is possible. It is also the class of algorithms that is mainly discussed in the DM literature. There are different ways of searching exhaustively all of which benefit from ways of pruning the search space so that only sub spaces are visited that have the potential of including solutions. But depending on the size of the search space and the selectivity of constraints and pruning techniques, the sub spaces to be traversed might still be too large.

A well-developed alternative is therefore the use of heuristic techniques for mining. In the second section, we discussed different approaches to heuristic search and highlighted the advantages and disadvantages of each. Especially in terms of local pattern mining, these techniques originate in the machine learning community. Heuristic pattern set mining techniques, on the other hand, have been introduced in data mining, mainly for the task of assembling predictive rules into decision lists. Using such techniques add the parameters of the search algorithm itself (such as selection criterion for the next solution to expand, beam size, etc) to the parameters that have to set for constraints. In this way heuristic approaches are much more susceptible to getting stuck in local optima. Additionally, once the number of solution to be returned increases, for instance in top-k or complete mining, the quality of each particular solution can be expected to decrease strongly. Unless all solutions are rather similar, which is also usually not in the end user's interest.

The size of the search space becomes less of an issue if there is not a large

number of elements that have to assembled into a solution, however. An alternative to the two afore-mentioned approaches, that would post-process patterns into pattern sets, lies in iterative mining. Iterative mining has been pioneered in the machine learning literature, in the form of sequential covering for rule learning and parallel covering for decision tree induction. In iterative mining each pattern is enumerates when it is needed, so to speak, keeping the size search space manageable. Iterative mining *is* usually a heuristic approach but one that gives better results since it allows to direct the search in such a way that individual patterns' shortcomings can be corrected. Even though iterative pattern set mining techniques have been wedded mainly to one particular approach local pattern mining, they form a general wrapper that allows us to instantiate the local mining step with different (exhaustive and heuristic) techniques.

Conclusion of Part I

This part of the thesis was used to lay out the foundations of our work. We began by defining the notions and settings discussed in the thesis in Chapter 1. Since pattern set mining is intricately linked to pattern mining, we specifically took care to formally state the problem of local pattern mining in detail in Section 1.1. By defining pattern languages, the properties of patterns and the constraints that arise from those properties, we gain the means to reason about the local pattern mining problem. We also discussed several applications in which local pattern mining and pattern set mining are used in Section 1.2 and related them exemplary to the two mining tasks

We then used Section 1.3 to discuss our main contribution in detail: a formal framework for pattern set mining. Starting with identifying the three types of sets usually used KDD, unordered, ordered and tree sets, we developed the vocabulary needed to discuss pattern sets and their properties. This allows the definition of constraints, and their characteristics, analogously to the well-developed local pattern mining setting. Using the terms we have introduced, we can analyze and improve existing approaches, and develop new techniques.

An important implication of our framework lies in the decomposition of local pattern mining and pattern set mining into separate and well-defined tasks. We used this insight to discuss algorithmic solutions in Chapter 2. Generally speaking, existing techniques can be differentiated into exhaustive and heuristic methods, and both classes have been applied to local pattern mining in data mining and machine learning. Existing pattern set mining techniques are almost exclusively heuristic, however. We identified two main paradigms of pattern set mining: as a post-processing step of a local pattern mining result and in an iterative fashion. Iterative mining will need to adhere to the heuristic approach, since the re-mining of patterns in each iteration leads to a too large search space. Post-processing deals with a limited number of patterns and therefore a smaller search space, on the other hand, and based on the properties discussed in Section 1.3, we proposed a novel exhaustive pattern set mining solution. Regarding the heuristic solutions originating in data mining, we characterized the general search strategies, and pointed out alternatives to existing solutions. Finally, with regard to iterative mining approaches, which originate in machine learning, we indicate that the integration with particular local pattern mining approaches is not a necessity, creating the possibility for new combinations of local pattern mining and pattern set mining techniques.

Part II

Pattern Set Mining as Post-processing

Overview of Part II

In the preceding section, we discussed a variety of algorithmic solutions to the pattern set mining, and connected to this, the local pattern mining problem. The first class of approaches to pattern set mining work as post-processing of the result set local pattern mining. There is a very clear advantage here in that local patterns are mined just once (usually in a rather efficient way due to the existing work in the field), saving computational complexity. The resulting set of patterns can then be subjected to rich constraints in selecting a subset that satisfies users' demands.

In the section on exhausting mining, Section 2.1, we claimed that exhaustive techniques for mining sets of local patterns that satisfy given constraints can be adapted to the pattern set mining task. This is a direct result of the framework we proposed for describing constraint-based pattern set mining and after the theoretical description, we aim to support those arguments by an experimental evaluation. The first section of Chapter 3 is therefore given to answering

1. Can an exhaustive mining algorithm be used in this manner, as we argued earlier?
2. Are constraints are useful in controlling the mining behavior?
3. Does the mining behavior differ from the local pattern mining setting?

Additionally, we propose two novel heuristic approaches to mining sets of unordered patterns in Section 3.2, which are are similar to feature selection techniques known from the machine learning literature. Using order-restriced hill-climbing and upper-bound ordered hill-climbing, we evaluate the effects that different choices (such as orders and measures) for these techniques have on the compactness and quality of the resulting pattern sets.

In Chapter 4, we consider ordered sets. On the one hand there are different ways of arriving at the order imposed on them, either during the mining process itself or after the result set has been computed. These orders have an effect on the quality of the resulting sets, as we will discuss in Section 4.1. Additionally, when using order-restricted hill-climbing, the used order becomes a relevant parameter of the pattern set mining step, as we show in detail in Sectioncut-the-crap. By replacing the order used in the only previously existing approach, CBA, and evaluating the differences, we gain insight into the effects of this parameter.

Chapter 3

Mining Unordered Sets by Exhaustive and Heuristic Search

We begin our evaluation of post-processing techniques by focussing on unordered pattern sets. Unordered pattern sets are attractive in that their composition and semantics are usually not tied to a particular task at hand. The relationship between patterns in such sets are often governed by pairwise constraints. As we have discussed in Section 1.3.5, these constraints have properties which have been employed in the design of local pattern mining algorithms to prune the search space and improve efficiency. Hence we claimed in Section 2.1 that those techniques can be adapted for the mining of pattern sets as well. The guarantees that these approaches give when it comes to the completeness and optimality of found solutions are clearly desirable.

In the first section (Section 3.1), we therefore evaluate the applicability of exhaustive mining to the task of pattern set mining as a second phase after exhaustive mining of local patterns. We point towards the similarities with the task of local pattern mining, especially in terms of pruning strategies, and algorithmic formulations. As we can show, constraints can be effectively enforced to make the mining process feasible. The similarities with local pattern mining unfortunately include the limitations of this approach as well, as the experimental evaluation shows.

The solution to this problem is to trade off completeness against efficiency, that is, employing a heuristic technique. In the second section (Section 3.2), we therefore choose a heuristic approach towards mining sets of patterns from existing local pattern mining output. Our main goal there is the mining of a *representative set* in terms of the semantics of the original pattern set. Given the right order and measures, order-restricted hill-climbing is an effective method towards this goal. We thus evaluate several choices for each aspect in terms of the size and usefulness of pattern sets mined in the context of concept learn-

ing. Reformulating those measures allows the calculation of upper bounds, and the abstraction from the order for mining. This leads to one particular solution (compared to order-specific ones from order-restricted hill-climbing) which compares well to the ones found by the other approach.

The two techniques evaluated in this chapter have been newly developed by us for this task, by applying a data mining approach, exhaustive mining under constraints, and a machine learning-inspired technique, similar to feature selection, to the problem. This chapter serves to introduce the two techniques in detail and motivate their use, in addition to showing their actual applicability to the task we address.

3.1 Exhaustive search under constraints

Upgrading local pattern mining techniques and extending them with pattern set constraints is a promising direction for the derivation of exhaustive pattern set mining techniques. It allows for the reuse of highly-developed enumeration and pruning techniques and leads to guaranteed optimal solutions. On the other hand, the entire field of pattern set mining stems from the fact that local pattern mining produces too many solutions. So, it has to be expected that this problem appears in exhaustive pattern set mining as well, alongside related problems such as a too large search space. Accordingly, this first section is concerned with evaluating the effectiveness and efficiency of exhaustive pattern set mining under constraints. This means that we evaluate whether monotone constraints can effectively prune the search space, giving the right thresholds. Additionally, we evaluate the effectiveness of top-k mining in forming sets of predictive patterns.

The general algorithm used for constrained pattern set mining in this section is the general two-phase algorithm which we described in Section 2.1.3 (Algorithm 7). We use two instantiations of this approach. In the first, the exhaustive search for local pattern mining satisfying certain support constraints is complemented by a second phase in which a conjunction of monotone and anti-monotone constraints is used to control the properties of mined pattern sets. Due to the reversed generality of pattern sets (large sets being more general), the monotone constraint is used to prune the pattern space. The anti-monotone constraint serves to control additional properties which cannot be enforced by pruning. In the second instantiation, the second phase takes the form of top-1 mining using a boundable function (and a monotone constraint) to return a single best pattern set, as an example application of the pattern set mining approach. In this case, both constraints can be leveraged for pruning purposes.

3.1.1 Experiment 1: Evaluating conjunctions *anti* $\wedge$ *mono*

For the first part of the experimental evaluation we mined local patterns on two datasets, and then executed the queries we gave as examples in Section 1.3.4. We reproduce those queries below for easier comprehension. The first data set is

the DTP aids antiviral database. On this data set we mined sequential patterns using the molecular feature miner MolFea (De Raedt and Kramer 2001), with a minimum support threshold on the *actives* of 27 and a maximum threshold on the *inactives* of 8. The resulting set of local patterns contains 287 patterns. This set was once used as is for the evaluation of Query 3.1.1 and Query 3.1.2.

$$sup(\mathbb{S}, Act) \geq \theta_{sup} \wedge max(red(\mathbb{S}, Mol)) \leq 1 \tag{3.1.1}$$

$$\begin{aligned} all(rep(\mathbb{S}, Act, Mol)) \geq 0.95 \wedge \\ max(red(\mathbb{S}, Mol)) \leq \theta_{red} \wedge \\ size(\mathbb{S}) \geq 2 \end{aligned} \tag{3.1.2}$$

For the evaluation of Query 3.1.3, a subset was selected in such a way that each *tid*-list occurred only once, leaving 29 patterns. This reduction is related to closed pattern mining but more selective since sequences might co-occur together which cannot be combined into a longer supersequence due to language constraints.

$$\begin{aligned} \mathbb{S} \preceq \{p_1\} \wedge \quad sup(\mathbb{S}, Act) \geq \theta_{s_i} \ \wedge \\ size(\mathbb{S}) \leq 20 \wedge \quad sup(\mathbb{S}, InAct) \leq \theta_{s_a} \end{aligned} \tag{3.1.3}$$

The second data set is the well-known UCI mushroom dataset. We mined maximal patterns both on the entire dataset and on each subset separately, using minimum support constraint $sup(p, \mathcal{D}) \geq 0.25$, finding 101 and 131 patterns, respectively. We then used those sets of local patterns for our experiments.

The two questions to be answered in our first experimental evaluation are:

Q3.1 How effective is the use of constraints in limiting the cardinality of pattern sets (facilitating further processing)?

Q3.2 How many pattern sets are returned and how efficient is exhaustive pattern set mining?

For this setting, we evaluated Queries 3.1.1, 3.1.2, and 3.1.3 with different thresholds on the patterns mined on the DTP aids antiviral database. Results are shown in Tables 3.1, 3.2, 3.3. All tables report the constraint thresholds, number of constrained sets returned, minimum and maximum cardinality of constrained sets, respectively.

In Table 3.1, θ_{sup} denotes the constraint placed on the minimum support of $\mathbb{S}$ on *Act*, in Table 3.2, θ_{red} denotes the maximum redundancy allowed. Only the values 0 and 11 are reported since any value below 11 behaves as in setting maximum redundancy to 0 and any value above 11 is obviously not practical. We also added a support constraint to Query 3.1.2, to additionally evaluate its effect; its threshold is again denoted by θ_{sup}:

$$\begin{aligned} all(rep(\mathbb{S}, Act, Mol)) \geq 0.95 \wedge \\ max(red(\mathbb{S}, Mol)) \leq \theta_{red} \wedge \\ size(\mathbb{S}) \geq 2 \wedge sup(\mathbb{S}, Mol) \geq \theta_{sup} \end{aligned} \tag{3.1.4}$$

Table 3.1: Query 3.1.1 evaluated on $\mathbf{L} = sup(p, Act) \geq 27$

θ_{sup}	$\lvert\mathbf{Th}\rvert$	min $\lvert\mathbb{S}\rvert$	max $\lvert\mathbb{S}\rvert$
50	8525392	2	5
100	8195886	3	5
150	2649152	5	5
160	42688	5	5
162	13920	5	5

Table 3.2: Query 3.1.4 evaluated on $\mathbf{L} = sup(p, Act) \geq 27$

θ_{red}	θ_{sup}	$\lvert\mathbf{Th}\rvert$	min $\lvert\mathbb{S}\rvert$	max $\lvert\mathbb{S}\rvert$
0	n/a	3690	2	2
0	40	180	2	2
0	50	90	2	2
0	55	32	2	2
0	57	0	n/a	n/a
11	n/a	776602	2	4
11	40	741028	2	4
11	50	134650	2	4
11	55	960	2	3
11	58	928	3	3
11	59	0	n/a	n/a

Regarding Query 3.1.1, we can see that the returned pattern sets are of low cardinality, facilitating human understanding and efficient use by a machine learning technique, without being explicitly constrained. Maximum redundancy obviously works in reducing the size of pattern sets. On the other hand is the cardinality of the entire theory $Th(\mathbf{L}, \mathcal{D}, \mathcal{C})$ far too large, numbering in the tens and even hundreds of thousands.

In evaluating Query 3.1.4, we see the same small cardinalities, and also how loosening the maximum redundancy threshold allows for pattern sets of larger cardinality. Furthermore, it can be observed how the interplay of a minimum support constraint and the succinct constraint on representativeness work to quickly reduce the cardinality of $Th(\mathbf{L}, \mathcal{D}, \mathcal{C})$ if θ_{sup} is raised.

Finally, in Table 3.3, θ_{s_i} and θ_{s_a} denote the maximum support on the *inactives* and minimum support on the *actives*, respectively.

Once again, the pattern sets are of low cardinality, lower than the 20 allowed by the size constraint. So on the one hand, returned set are never very large, facilitating the understanding of the results. On the other hand, the amount of pattern sets returned is high, even for restrictive constraint settings.

Additionally, as can be seen, selecting the right thresholds for the constraints is a non-trivial task. Sometimes, changing support by 1 means that no pattern set is returned at all. These are well-known phenomena that hold for local pattern mining as well. The fact that, as we expected, they are mirrored in the

Table 3.3: Query 3.1.3 evaluated on $\mathbf{L} = sup(p, Act) \geq 27$, unique *tid*-lists

θ_{s_i}	θ_{s_a}	$\lvert\mathbf{Th}\rvert$	min $\lvert\mathbb{S}\rvert$	max $\lvert\mathbb{S}\rvert$
10	30	7678	2	13
10	50	6040	2	13
10	100	2	5	6
20	30	706975	2	17
20	50	703904	2	17
20	100	242377	4	17
20	130	2417	5	13
20	137	541	6	13
20	138	0	n/a	n/a

Table 3.4: Mushroom dataset, maximal sets mined on complete dataset $max(red(\mathbb{S}, All)) < 1$, varying support

	$\lvert\mathbf{Th}\rvert$	min $\lvert\mathbb{S}\rvert$	max $\lvert\mathbb{S}\rvert$
$sup(\mathbb{S}, All) > 0$	923	1	3
$sup(\mathbb{S}, All) > 3000$	823	1	3
$sup(\mathbb{S}, E) > 3000$	245	2	3
$sup(\mathbb{S}, P) > 3000$	171	2	3

task of constrained pattern set mining provides additional support to our claim that the two tasks are structurally similar.

An interesting result is that using the patterns without ad-hoc removal of non-unique *tid*-lists allows for a larger minimum support on *Act* than if those patterns are removed. This shows that the task of removing non-unique *tid*-lists is itself essentially a constrained pattern set mining task, one that we will address to a certain degree in the next section.

In the case of patterns mined on the mushroom dataset, we constrained the maximum redundancy to 0 and evaluated the effect of setting a minimum support threshold of 3000 on the entire dataset, and each of the subsets. The results for these experiments are shown in Tables 3.4 and 3.5.

Again, it can be seen that pattern sets are of small cardinality, making it easy for the user to comprehend the entire set. On the other hand, even setting relatively strong thresholds (considering that each subset consists of roughly

Table 3.5: Mushroom dataset, maximal sets mined on subsets $max(red(\mathbb{S}, All)) < 1$, varying support

	$\lvert\mathbf{Th}\rvert$	min $\lvert\mathbb{S}\rvert$	max $\lvert\mathbb{S}\rvert$
$sup(\mathbb{S}, All) > 0$	23647	1	6
$sup(\mathbb{S}, All) > 3000$	21227	2	6
$sup(\mathbb{S}, E) > 3000$	1063	2	6
$sup(\mathbb{S}, P) > 3000$	5046	3	6

4000 instances) still leads to a large number of pattern sets returned.

As we have seen, the answer to Question 3.1 is that using a monotone constraint (redundancy) with the right threshold is very effective in limiting the cardinality of patterns. This is the fact even *without* an explicit constraint on their size.

Unfortunately, we have to answer Question 3.2 to the effect that exhaustive pattern set mining behaves very similar to exhaustive local pattern mining. This includes the regrettable effect of typically producing a far larger theory than humans can peruse. On the other hand, the cardinality of returned pattern sets is relatively small, meaning that every single set can be processed by humans and algorithms efficiently.

On the positive side this means that applying the lessons learned in local pattern mining regarding condensed representations, different enumeration (and ordering) strategies, and techniques for top-k-mining will be applicable to the pattern set mining task as well.

3.1.2 Experiment 2: Classifier construction

CBA first mines so-called *class association rules*, rules whose target pattern is a class label to realize classification. It uses a level-wise exhaustive algorithm (Apriori) for mining those rules according to certain minimum support and minimum confidence constraints. In a second phase, it uses order-restricted hill-climbing to mine a pattern set which is then used as the classifier. The classification strategy is that of a *decision list*, meaning that rules are checked against an unlabeled instance in order and the first one matching used to predict the class label.

The exhaustive alternative lies in using a constraint like Query 3.1.5. It allows us to evaluate pattern with regard to the other patterns in the set, giving up the order-restriction.

$$\max(red(\mathbb{S}, D_{train})) \leq \theta_{red} \wedge \arg_1 \max_{\mathbb{S} \in 2^{\mathbf{L}}} acc(\mathbb{S}, \mathcal{D}_{train}) \geq \theta_{acc} \qquad (3.1.5)$$

If the order is given up, the overlap between rules should be small in order to reduce conflicting predictions for unseen data, which leads to the first constraint. At the same time, maximally accurate rule set on the training data should be mined.

The questions we consider in this experiment are:

Q3.3 Does the exhaustive strategy lead to smaller and/or more effective classifiers?

Q3.4 How large is the computational effort needed to construct a classifier by exhaustive means?

To illustrate this, we set up the following small experiment: using CBA, we mined class association rules on ten folds of the UCI balance-scale dataset. On average 98.8 rules ($\pm$17.38) were found. When randomly permuting the order

Table 3.6: Comparing CBA's classifier with constrained set mining result

	Size set	Acc_{train}	Acc_{test}
CBA	17.4 ± 5.1	0.8443 ± 0.019	0.7918 ± 0.0459
CSM	4.9 ± 1.66	0.8443 ± 0.019	0.7918 ± 0.0459

of equal rules and using CBA's post-processing step on the resulting rule sets, classifiers of different size and accuracy on the training sets are constructed.

Then we chose one permutation and additionally ran Query 3.1.5 on that permutation to select a pattern set used as final classifier. Average and standard deviation for the size of the rule set selected by the post-processing step, accuracy on the training and the test data are shown in Table 3.6 for CBA and our constrained pattern set mining (CSM) approach in the first and second row, respectively. Maximum redundancy was set to 5 and the accuracy threshold to 0.5.

In answering Question 3.3, we find that while both approaches construct classifiers of the same quality, the one constructed by constrained pattern set mining is considerably compacter. Visual inspection of the rule sets mined by our approach shows that often rules are selected that are not included in CBA's solution. Specifically, quite often, the highest-ranked rule is ignored by our approach while CBA's post-processing will always include this one. Additionally, on several folds, the constrained pattern set did not include a single rule with confidence 1 without decreasing the accuracy of the resulting classifier.

While mining for the constrained pattern sets, on average 18.6 levels were traversed (± 2.319), far from the search space that an exhaustive search without pruning of on average 99 rules would need to consider. This means that the computational cost of using exhaustive search for constructing such a classifier is manageable if the right constraints are used, answering Question 3.4.

3.1.3 Conclusions

Extending the work on exhaustive pattern set mining that has existed so far, we proposed and evaluated a general exhaustive miner that can enforce arbitrary constraints. This contrasts with existing work which always includes a size-constraint to control computational complexity. As our experiments show, a general exhaustive miner can be a useful tool but can be quickly overwhelmed by the combinatorial explosion arising from a large set of local patterns that has to be post-processed. Since the subgoal of the KDD process that we aim to address with our work is understandability, mining a large amount of pattern sets is questionable. The alternative lies in heuristic techniques that optimize only one (or few) pattern sets.

3.2 Heuristic search

As we have demonstrated in the preceding section, exhaustive search is an effective way of mining pattern sets satisfying specified constraints. Unfortunately, this mining operation also needs a large amount of computational resources and there are typically too many pattern sets that satisfy the constraints. So in a sense, exhaustive pattern set mining reintroduces the problem it is supposed to solve, namely the reduction of a large local pattern mining result set to a smaller set that is meaningful to humans and usable by algorithms.

Especially when it comes to further algorithmic processing, a single set of patterns usually suffices. Of course it is possible to perform top-1 mining to select such a set but depending on the effectiveness of pruning, the computational effort of the exhaustive search is still high. Since top-1 mining needs a measure Φ to be optimized for the pattern set, hill-climbing is an alternative that suggests itself. Two important aspects are once again that the pattern set is *representative* of the entire local pattern mining result, and that redundancy among patterns is small.

The contents of this section, published in (Bringmann and Zimmermann 2007) and (Bringmann and Zimmermann 2009), are the result of joint work with Björn Bringmann at the Katholieke Universiteit Leuven.

3.2.1 Notions

For exhaustive search, we identified the maximum redundancy constraint as a powerful pruning constraint in exhaustive pattern set mining. We also saw, however, that setting effective thresholds is not straight-forward. Additionally, maximum redundancy fails to prevent a particular type of redundancy that is of interest for further algorithmic processing when learning concepts for instance:

Example 3.2.1. *Consider two patterns p_1, p_2 with the following relationship:*

$$(match(p_1, \mathcal{D}) = 1 \Leftrightarrow match(p_2, \mathcal{D}) = 0) \wedge (match(p_1, \mathcal{D}) = 0 \Leftrightarrow match(p_2, \mathcal{D}) = 1)$$

Those patterns will be selected for inclusion in the same pattern set by a maximum redundancy constraint. To a concept learning algorithm that uses the patterns as features either one is equally good and having both does not add any information. Similarly, while this particular relationship might be interesting to a human user, the information contained in a pattern set is not increased by having both patterns.

We therefore use the concept of the *equivalence relation* which we introduced in Section 1.2.3:

Definition 3.2.1. Pattern Set-induced Equivalence Relation *Given a pattern set $\mathbb{S}$ we define an equivalence relation $\sim_{\mathbb{S}}$ on the set $\mathcal{D}$ of instances as*

$$\sim_{\mathbb{S}} = \{(e_1, e_2) \in \mathcal{D} \times \mathcal{D} \mid \forall p \in \mathbb{S}\ match(p, e_1) = match(p, e_2)\}$$

Thus, two instances are considered to be equivalent under $\mathbb{S}$ if they share exactly the same patterns. Since every equivalence relation induces a partition, pattern sets can be compared in terms of the partitions they induce.

To get an intuition why partitions are central to our technique, consider the following: For a machine learning algorithm, a subset $\mathbb{S}^*$ that induces the same partition as $\mathbb{S}$ will be of the same usefulness as the complete set since the separability of instances is equally well possible. Taking into account the actual syntactical composition and support of the patterns in the selection measure requires measures that are specifically bound to certain types of pattern languages. Since our approach aims at a more general applicability, namely all patterns that can be defined as absent or present, further details on syntactical composition or similar relations are *not* of interest here.

For a human user with elaborate knowledge about the domain the situation might be somewhat different. But by defining the total order in which to process patterns the user can control, to some extent, which patterns are considered for selection (for the order-restricted hill-climbing search). Furthermore, a domain expert can also incorporate his knowledge into the process by selecting a set of patterns and using the algorithm to complete this set with patterns of value according to the method employed.

3.2.2 Order-restricted pattern set mining

As an illustration of how the addition of patterns can change the equivalence relation, and, therefore, the partition, consider Figure 3.1. The left-hand side shows a small snapshot of $\mathcal{D}$. As can be clearly seen on the left-hand side, p_3's presence depends on the presence of both p_1 **and** p_2, and p_4's presence on the presence of p_1 and *absence* of p_2. The right-hand side shows the partition induced by the first three patterns on $\mathcal{D}$, and shows that p_3's dependency is mirrored in the way $\mathcal{D}$ is split into cells.

This leads to the basic redundancy-decreasing constraint we employ:

$$\mathcal{C}(\mathbb{S}, \mathcal{D}) \equiv \neg\exists p \in \mathbb{S} : |\mathcal{D}/\sim_{\mathbb{S}}| = |\mathcal{D}/\sim_{\mathbb{S}\setminus\{p\}}|$$

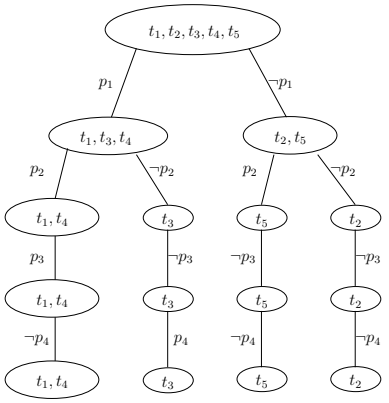
This constraint is used to reject the inclusion of patterns that are redundant with regard to the rest of the set during the mining operation. Note that this rejection means that the $\mathbb{S}^*$ created by this selection criterion will induce the *same* partition as the *whole* $\mathbb{S}$.

The maximum size of a pattern set that could be induced by this constraint is $|\mathcal{D}| - 1$. The minimum number of patterns *needed* to induce a partition of size $\mathcal{D}/\sim_{\mathbb{S}^*}$, however, is $\log_2 |\mathcal{D}/\sim_{\mathbb{S}^*}|$. While this number is hardly reachable, the goal still is to mine small pattern sets, and therefore an additional measure is needed. Ideally, this measure would trade off size of the pattern set against the quality of the partition. As there is no straight-forward choice, there are two ways of decoupling the two properties:

1. One solution lies in restricting the size of $\mathbb{S}^*$ to a small integer and exhaustively enumerate and evaluate all pattern sets of that size, the approach chosen in (Knobbe and Ho 2006b).

	t_1	t_2	t_3	t_4	t_5
p_1	x		x	x	
p_2	x			x	x
p_3	x			x	
p_4			x		

(a) Transactions and pattern occurrence



(b) Partition tree

Figure 3.1: Partitions induced by patterns. Four binary patterns can induce at most 16 cells. This combination of patterns and instances yields only four cells, however.

2. Alternatively, when using heuristic mining, a measure Φ can be defined:

$$\Phi(\mathcal{D}, \mathbb{S}^*, p) \rightarrow [0, 1] \subset \mathbb{R}$$

which measures the strength of the contribution an individual pattern makes to an existing pattern set. If Φ is constructed in such a way that the patterns' contribution decreases with increasing pattern set size, it can be combined with a threshold to produce small pattern sets that create good partitions.

We will discuss several ways of measuring this contribution in the following section, and afterwards propose orders to be used in order-restriced hill-climbing.

3.2.3 Quality measures

To give an idea of the form that measures should take that quantify a pattern's contribution to the set, we discuss three potential instantiations here. To give some more motivation for each of them, we will attempt to give some "meaning" to each selection measure used. Two of them, Φ_C and Φ_I in fact "outsource" part of the measure calculation to an outside algorithm. This is enabled by the reduction of computational complexity due to the order-restriction.

Partition size quotient Φ_Q

While rejection of patterns that do not change the partition will effectively cut down on the number of patterns retained already, there is the possibility that adding a pattern will affect only a few cells. While this may be acceptable in early steps of the selection process when not many patterns are used and only

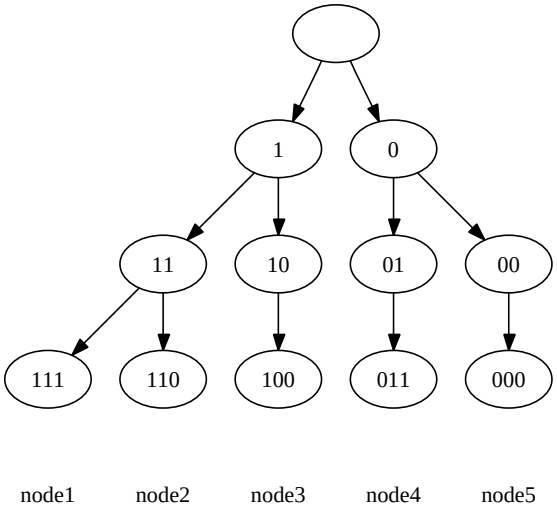


Figure 3.2: Bitstrings denoting presence (1) and absence (0) of patterns

few cells formed, in later steps this corresponds to only a small gain in new information. Let us assume for instance that exactly one of the existing cells is split into two sub-cells when a new pattern is added. This means that the total number of cells is increased by one. Depending on the number of already existing cells, this e.g. corresponds to 33% for two cells, but only 0.9% for 100 cells.

One way of measuring uses

$$\Phi_Q(\mathcal{D}, \mathbb{S}^*, p) = 1 - \frac{|\mathcal{D}/\sim_{\mathbb{S}^*}|}{|\mathcal{D}/\sim_{(\mathbb{S}^* \cup \{p\})}|}.$$

As the example given above shows, this measure decreases with an increasing number of cells. We can now define a threshold on what we perceive to be an acceptable increase in the number of cells and use it for additional pattern selection. The main advantage of this criterion is that it is easy to evaluate. A possible disadvantage is that focusing solely on the number of cells without considering which cells are split and which instances are contained in the new sub-cells might not be sufficient.

Agglomerative clustering Φ_C

To alleviate this potential problem, one might use an agglomerative clusterer that combines some of the new sub-cells until the old number of cells is reached. Thus, a clustering algorithm can be denoted as a function $\mathcal{F}(\mathcal{D}, k)$ that returns a partition $\mathbb{P}$ of $\mathcal{D}$ consisting of k cells.

Let us assume that, as in the section before, the addition of a new pattern leads to a split of an existing cell into two sub-cells, as shown in Figure 3.2.

These (*node1* and *node2*) have a dissimilarity of 1 according to the Manhattan distance since they agree in all patterns except the new one. By the same argument the parent cell had a distance of at least 1 to the other cells at its level. Thus the sub-cell in which the new pattern is present will have a distance of at least 2 to all cells where this pattern is absent (except to its sibling), and vice versa. In this case *node1* has Manhattan distance of ≥ 2 to *node3* and *node5*. Non-split cells can have a smaller dissimilarity to non-sibling cells than to siblings, depending on the effect of the new pattern, as is the case for *node3* and *node5*. Similarly, ignoring the sibling relation between *node1* and *node2*, *node1* is more similar to *node4* than to any other node, and *node2* to *node3*.

An agglomerative clusterer will combine two cells from the new partition into one cell since then the old number of cells is reached again. Given the effects described above, there will very likely be change that can be measured using the *Rand* index, affected by the new pattern over the entire partition, even unchanged cells. For a bigger increase in the number of cells this change can be expected to be even more pronounced.

The *Rand* index considers two partitions and considers all pairs of instances. It measures whether the relationship in those pairs (whether they belong to the same or different clusters) is retained. The *Rand* index is defined as follows: assume two partitions P, P'. For each pair of instances e_i, e_j, two decision variables can be defined - c_{ij} which is set to 1 if the two instances end up in the *same* cell in both P and P', 0 otherwise, and $d_{ij} = 1$ if the two instances are assigned to *different* cells in both partitions, 0 otherwise.
The *Rand* index is then:

$$Rand(P, P') = \frac{2 \cdot \sum_{i=1}^{|\mathcal{D}|-1} \sum_{j=i+1}^{n} (c_{ij} + d_{ij})}{|\mathcal{D}| \cdot (|\mathcal{D}| - 1)}$$

We define

$$\Phi_C(\mathcal{D}, \mathbb{S}^*, p) = 1 - Rand(\mathcal{D}/\sim_{\mathbb{S}^*}, \mathcal{F}(\mathcal{D}/\sim_{\mathbb{S}^* \cup p},\ |\mathcal{D}/\sim_{\mathbb{S}^*}|))$$

and set a threshold θ that quantifies what we consider the minimal acceptable number of changes for a pattern to be chosen. Once again, this is a measure that decreases with increasing number of cells.

This technique will take longer to evaluate than the one shown before since the clustering process needs at least quadratic running time and for the *Rand*-index $\frac{1}{2} \cdot n(n-1)$ pairwise decisions have to be evaluated. It does have the advantage of using information about the size and composition of the cells and not only about their number though.

Please note that Φ_C will always yield 0 if $\mathbb{S}^*$ is empty, since the clustering would then have to create a partition with *one* cell. Thus its *Rand*-index is zero. To overcome this problem the measure will return 1 for any pattern that can create two cells if $\mathbb{S}^*$ is empty.

Inference of patterns Φ_I

The first selection measure we showed strictly evaluates whether patterns can be described by a combination of others while the second one evaluates the effect of combining instances that differ in only one pattern. A third option uses a rule-based machine learning technique to evaluate the possibility of predicting the presence/absence of a pattern based on the presence of previously chosen patterns. While this will never lead to a perfect model[1], a pattern whose presence or absence is correctly predicted on the majority of instances can be considered as not adding much information.

Given a pattern set $\mathbb{S}^* = \{p_1, \ldots, p_k\}$, a new candidate pattern p and the database $\mathcal{D}$, we identify each transaction e_i with its binary feature vector $\overrightarrow{f_{\mathbb{S}^*}}(e_i) = (match(p_1, e_i), \ldots, match(p_k, e_i))$ and label it with $c(e_i) = match(p, e_i)$. We use a learner[2] to induce a hypothesis $h : X \mapsto \{0, 1\}$ where $X = \{\overrightarrow{f_{\mathbb{S}^*}}(e) \mid e \in \mathcal{D}\}$ and define the measure

$$\Phi_I(\mathcal{D}, \mathbb{S}^*, p) = 1 - \frac{|\{e_i : h(\overrightarrow{f_{\mathbb{S}^*}}(e_i)) = c(e_i)\}|}{|\mathcal{D}|}.$$

Note that in this case *all* instances are represented as binary vectors including duplicates w.r.t. the feature vectors. This means that a feature having only marginal effect on large parts of the instance space will be predicted with high accuracy while a feature that for instance splits the largest cell in half will have far less accuracy, thus having a better chance of being chosen. Again, given that cells become smaller during the mining process, this measure decreases as the number of cells increases.

3.2.4 Ordering relations

As we mentioned before, the first technique we present performs order-restriced hill-climbing and therefore needs orders on the patterns. For the experimental evaluation (which we performed on patterns from $\mathcal{L}_\mathcal{I}$), we defined the following orders:

- $\preceq_{s\uparrow} : p_1 \lhd_{s\uparrow} p_2 \Leftrightarrow sup(p_1, \mathcal{D}) < sup(p_2, \mathcal{D})$
- $\preceq_{s\downarrow} : p_1 \lhd_{s\downarrow} p_2 \Leftrightarrow sup(p_1, \mathcal{D}) > sup(p_2, \mathcal{D})$
- $\preceq_{l\uparrow} : p_1 \lhd_{l\uparrow} p_2 \Leftrightarrow |p_1| < |p_2|$
- $\preceq_{l\downarrow} : p_1 \lhd_{l\downarrow} p_2 \Leftrightarrow |p_1| > |p_2|$

While the two orders referring to the cardinality of the itemsets are tailored to this particular pattern domain, similar orders can be defined for other languages. In the same vein, orders can be used to incorporate background knowledge into the mining process.

[1] Due to the rejection of patterns that do not effect a change in the partition.
[2] The J48 implementation of WEKA

By using these orders in an order-restricted hill-climber (cf. Section 2.2.2) and employing any of the measures described above, the mining of small non-redundant sets of patterns is possible. We call this heuristic miner BOUNCER: like a doorman at a venue it considers each candidate *once*, accepting or rejecting it and never revisiting its decisions. This makes search space traversal reasonably fast, allowing for the outsourcing of quality evaluation.

3.2.5 Upper-bound ordered pattern set mining

In the BOUNCER algorithm, a pattern that just exceeds the threshold will be chosen, neglecting the possibility of any other pattern later in the sequence reaching a much higher score, and thus introducing a more desireable split. To address this issue, in each selection step each pattern has to be evalulated in order to select the pattern with the highest score. However, except for Φ_Q, evaluating the score of all patterns in every selection step introduces a much higher computational complexity.

Much of this complexity arises from the use of a classification/clustering algorithm in two of the selection measures. The classification algorithm used in Φ_I allows the rejection of patterns that only split off small cells, and usage of a sophisticated algorithm (such as an SVM) or evaluation schemes such as cross-validation might correct over-fitting effects. The use of an unpruned classifier evaluated on training-data – the most efficient method – equates to simple counting.

Similarly, while we originally chose the clustering technique to introduce an element of chance to offset the ordering effect, the question is whether this random effect significantly improves the solution. Furthermore, given the systematic dissimilarity of cells, a maximally different clustering could be derived from analysis of the partition-tree.

Therefore, a possible solution to the effects of the sequential processing lies in foregoing the algorithmic components within our measures. Instead, we propose new formulations that allow the calculation of upper bounds for patterns, thus allowing for pruning, and ideally, enough efficiency-gain to compensate for the repeated evaluation of patterns.

3.2.6 Reformulating the quality measures

In the following sections, we will introduce new, semantically equivalent, formulations of the measures used in BOUNCER to allow for upper bounding the contribution of candidate patterns. We follow this by formulating the respective upper bounds.

Reformulating the partition size quotient Φ_Q

The first measure, Φ_Q, quantifies to what degree a new pattern fulfills its potential to make the existing partition more fine-grained. It is formulated as

$$\Phi_Q(\mathcal{D}, \mathbb{S}^*, p) = 1 - \frac{|\mathcal{D}/\sim_{\mathbb{S}^*}|}{|\mathcal{D}/\sim_{(\mathbb{S}^* \cup \{p\})}|}$$

In this formulation, it is measured how large the amount of unchanged cells is. A reformulation based on the same idea would be

$$\Phi_{Q^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{|\mathcal{D}/\sim_{(\mathbb{S}^* \cup \{p\})}| - |\mathcal{D}/\sim_{\mathbb{S}^*}|}{2 \cdot |\mathcal{D}/\sim_{\mathbb{S}^*}|}.$$

Φ_{Q^+} returns the fraction of newly introduced cells by the candidate pattern over the maximal amount of cells that could result from introducing a new pattern. Thus, if the candidate pattern does not introduce a new split, Φ_{Q^+} equals 0. If a pattern splits *every* cell, doubling the total number of cells, $|\mathcal{D}/\sim_{(\mathbb{S}^* \cup \{p\})}| = 2 \cdot |\mathcal{D}/\sim_{\mathbb{S}^*}|$, and the score therefore $\frac{1}{2}$.

Removing the element of chance from Φ_C

The way Φ_C is formulated, the cells formed after applying the candidate pattern are recombined using an agglomerative clusterer and then compared to the old partition using the *Rand*-index. Since a trivial approach to this would simply recover the old partition, care is taken that non-siblings are combined if possible, thus ensuring the introduction of change. The same could be effected by comparing the old and the new partition via the *Rand*-index. This would remove the element of chance, and reduce the computational complexity – resulting in a lower execution time. Analogous to the measures introduced earlier we define this *Rand*-based measure as:

$$\Phi_{C^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{2}{|\mathcal{D}| \cdot (|\mathcal{D}| - 1)} \cdot \sum_{[e] \in \mathcal{D}/\sim_{\mathbb{S}^*}} |cov(p, [e])| \cdot |\overline{cov(p, [e])}|$$

This new measure is based on the observation that cells can be considered separately. For each cell $[e]$, the number of pairs that will be split is just the product of the number of instances that are covered by the candidate pattern and those which are not. Since splits are never reversed during the selection process, instances that are in different cells can never be merged into one cell and hence do not contribute to the *Rand*-index. The summation of these products over all cells is normalized by the maximum number of pairs that can split which is $\frac{|\mathcal{D}| \cdot (|\mathcal{D}|-1)}{2}$, as mentioned before.

Please note that the *Rand*-index usually counts the number of pairs that are *not* split, whereas here we count the number of pairs that will be split, thus *inverting* the *Rand*-index.

Formulating Φ_I in closed form

Currently, a classifier is employed to find the score a pattern achieves, in the form of:

$$\Phi_I(\mathcal{D}, \mathbb{S}^*, p) = 1 - \frac{|\{t_i \in \mathcal{D} : h(\overrightarrow{f_{\mathbb{S}^*}}(t_i)) = c(t_i)\}|}{|\mathcal{D}|}$$

which is equivalent to $1 - accuracy$ of the hypothesis induced by the learning algorithm. The learner we employed for Φ_I uses a decision tree mechanism, which means that it learns a set of mutually exclusive conjunctive rules. In addition we used the tree unpruned and evaluated it on the training data such that actually the (observed) probability $\mathcal{P}(p \mid p_1, \ldots, p_k)$ is measured. This probably can also be found by counting the number of occurrences of p on each cell.

More formally:

$$\Phi_{I^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{1}{|\mathcal{D}|} \sum_{[e] \in \mathcal{D}/\sim_{\mathbb{S}^*}} \min\left\{|cov(p, [e])|, |\overline{cov(p, [e])}|\right\}$$

This definition equals the *minimum error* a classifier can achieve on the training set $\mathcal{D}$ using all features contained in $\mathbb{S}^*$. Thus $\Phi_{I^+} = \Phi_I$ if the learning algorithm employed in Φ_I builds an optimal model for the supplied dataset $\mathcal{D}$. This reformulation strongly reduces the complexity and thus can be used in an algorithm that does not use order-restricted hill-climbing.

This algorithm, which we will refer to as PICKER, will make more informed decisions than BOUNCER, since it greedily "picks" the locally best pattern for inclusion. Depending on the number of patterns to select from, even the more efficient measure calculations can be too expensive, however.

3.2.7 Using bounds to reduce complexity

The new formulations we introduced, have been chosen in such a way that it is possible to bound the contribution that each pattern can bring to a pattern set. This allows us to do upper-bound ordered hill-climbing and turn the simple hill-climber BOUNCER into the greedy algorithm PICKER*. As we have explained before, in upper-bound ordered hill-climbing, patterns are considered as candidates for extending the pattern set in the order based on their upper bounds, with highest upper bounds coming first. Every time a pattern's p_i score exceeds the score of the best pattern p_j encountered so far ($\forall j < i : \Phi(\mathcal{D}, \mathbb{S}^*, p_i) > \Phi(\mathcal{D}, \mathbb{S}^*, p_j)$), this pattern is considered the candidate pattern and its upper bound *calculated*. Once the upper bounds Φ^* of further candidate patterns falls below the score the current candidate for extension p_c achieves ($\Phi(p_c) > \Phi^*(p_i)$ for a given i), those patterns do not need to be evaluated in the data base. However, the current bound needs to be *updated* since the newly introduced split can change the effect of later patterns. This update is performed in constant time.

A bound for Φ_{Q^+}

Denoting the number of splits a candidate pattern would introduce in step k as $splits_k(p)$, the bound for the reformulated quotient can be expressed as

$$\Phi^*_{Q^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{2^{|\mathbb{S}^*|-k} \cdot splits_k(p)}{2 \cdot |\mathcal{D}/\sim_{\mathbb{S}^*}|}$$

In order to show that this is indeed an upper bound, we need to prove the following proposition:

Theorem 3.2.1. *Given pattern sets $\mathbb{S}, \mathbb{X}$, pattern p:*

$$\Phi_{Q^+}(\mathcal{D}, \mathbb{S}^* \cup \mathbb{X}, p) \leq \Phi^*_{Q^+}(\mathcal{D}, \mathbb{S}^* \cup \mathbb{X}, p)$$

Proof. We need to show that for any pattern set $\mathbb{X}$

$$\frac{|\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}\cup\{p\}}| - |\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}}|}{2 \cdot |\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}}|} \leq \frac{2^{|\mathbb{S}^*\cup\mathbb{X}|-k} \cdot (|\mathcal{D}/\sim_{\mathbb{S}^*\cup\{p\}}| - |\mathcal{D}/\sim_{\mathbb{S}^*}|)}{2 \cdot |\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}}|}$$

Knowing $|\mathbb{S}^*| = k$ and $\mathbb{S}^* \cap \mathbb{X} = \emptyset$ we have

$$|\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}\cup\{p\}}| - |\mathcal{D}/\sim_{\mathbb{S}^*\cup\mathbb{X}}| \leq 2^{|\mathbb{X}|} \cdot (|\mathcal{D}/\sim_{\mathbb{S}^*\cup\{p\}}| - |\mathcal{D}/\sim_{\mathbb{S}^*}|)$$

since the maximal number of splits that $\mathbb{X}$ could have introduced is $2^{|\mathbb{X}|}$ and any blocks p splits are among those blocks in the best case.

Given that any block can be split into maximally two new blocks by a single pattern, we can express the difference between partitions in terms of the effect of a pattern on individual blocks. This allows us to reformulate the inequality as:

$$\sum_{[e]\in\mathcal{D}/\sim_{\mathbb{S}^*}} \sum_{[e']\in[e]/\sim_{\mathbb{X}}} (|[e']/\sim_{\{p\}}| - 1) \leq 2^{|\mathbb{X}|} \cdot \sum_{[e]\in\mathcal{D}/\sim_{\mathbb{S}^*}} |[e]/\sim_{\{p\}}| - 1$$

Again it is sufficient to show the inequality for each of the summand due to monotonicity of the sum. Thus $\forall [e] \in \mathcal{D}/\sim_{\mathbb{S}^*}$

$$\sum_{[e']\in[e]/\sim_{\mathbb{X}}} (|[e']/\sim_{\{p\}}| - 1) \leq 2^{|\mathbb{X}|}(|[e]/\sim_{\{p\}}| - 1)$$

We now have to consider two cases:

1. If $\forall [e'] \in [e]/\sim_{\mathbb{X}} \quad |[e']/\sim_{\{p\}}| = 1 \Longrightarrow |[e]/\sim_{\{p\}}| = 1$ then

   $$2^{|\mathbb{X}|} \cdot (1-1) \geq \sum_{[e']\in[e]/\sim_{\mathbb{X}}} (1-1) \quad \checkmark$$

   In this case, any block that is introduced by $\mathbb{X}$ is *not* split by p. The inequality holds in this case.

2. If $\exists [e'] \in [e]/\sim_{\mathbb{X}} \quad |[e']/\sim_{\{p\}}| = 2 \Longrightarrow |[e]/\sim_{\{p\}}| = 2$ then

$$2^{|\mathbb{X}|} \cdot (2-1) \geq \sum_{[e'] \in [e]/\sim_{\mathbb{X}}} (2-1)$$
$$2^{|\mathbb{X}|} \geq |[e]/\sim_{\mathbb{X}}| \quad \checkmark$$

In the second case, there exist blocks that are split by p and the inequality holds as well.

As both equations are true, this means that the upper bound for each single summand holds and the claim follows. □

A bound for Φ_{C^+}

The calculated *Rand* serves as an upper bound.

$$\Phi^*_{C^+}(\mathcal{D}, \mathbb{S}^*, p) = \Phi_{C^+}(\mathcal{D}, \mathbb{S}^*, p)$$

Even though this looks surprisingly simple, it is a stronger upper bound than the upper bound for the quotient, which increases in each step whereas this upper bound does not increase and hence needs no update. It is also superior in that it will allow actual pruning of patterns whose *Rand* index does not clear the threshold anymore.

To show that this is actually an upper bound we need to prove the following proposition:

Theorem 3.2.2. *Given pattern sets $\mathbb{S}, \mathbb{X}$, pattern p:*

$$\Phi_{C^+}(\mathcal{D}, \mathbb{S}^* \cup \mathbb{X}, p) \leq \Phi_{C^+}(\mathcal{D}, \mathbb{S}^*, p)$$

Proof. We will prove the claim using induction by showing that:

$$\forall p, p_x \in \mathbb{X} : \Phi_{C^+}(\mathcal{D}, \mathbb{S}^*, p) \geq \Phi_{C^+}(\mathcal{D}, \mathbb{S}^* \cup \{p_x\}, p).$$

Recall that Φ_{C^+} (and thus $\Phi^*_{C^+}$) is defined as a normalised sum:

$$\Phi_{C^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{2}{|\mathcal{D}| \cdot (|\mathcal{D}| - 1)} \cdot \sum_{[e] \in \mathcal{D}/\sim_{\mathbb{S}^*}} |cov(p, [e])| \cdot |\overline{cov(p, [e])}|.$$

Hence it is sufficient to show that each summand is monotonically decreasing with the introduction of an additional pattern p_x or in other words, that we can consider the effect of p_x *per block*. That is, $\forall p, p_x \in \mathbb{X}$ and $\forall [e] \in \mathcal{D}/\sim_{\mathbb{S}^*}$:

$$|cov(p, [e])| \cdot |\overline{cov(p, [e])}| \geq \sum_{[e'] \in [e]/\sim_{\{p_x\}}} |cov(p, [e'])| \cdot |\overline{cov(p, [e'])}|.$$

Note that $|[e]/\sim_{\{p_x\}}| \leq 2$. The two patterns p, p_x can induce at most $2^2 = 4$ blocks from an existing block. Let $[e_1], [e_2], [e_3], [e_4] \in [e]/\sim_{\{p_x,p\}}$ be these four pairwise disjunct blocks such that $cov(p, [e]) = cov(p, [e_1] \cup [e_3]) = |[e_1] \cup [e_3]|$ and $cov(p_x, [e_1] \cup [e_2]) = |[e_1] \cup [e_2]|$. This means that $[e_1]$ is the block in which both p and p_x occur, $[e_2]$ the one in which p does not occur and p_x does etc. Therefore we can make the sum explicit and reformulate the inequality above as

$$\begin{array}{lrcl} & |[e_1] \cup [e_3]| \cdot |[e_2] \cup [e_4]| & \geq & |[e_1]| \cdot |[e_2]| + |[e_3]| \cdot |[e_4]| \\ \Leftrightarrow & (|[e_1]| + |[e_3]|) \cdot (|[e_2]| + |[e_4]|) & \geq & |[e_1]| \cdot |[e_2]| + |[e_3]| \cdot |[e_4]| \\ \Leftrightarrow & |[e_1]| \cdot |[e_2]| + |[e_1]| \cdot |[e_4]| \quad + & & \\ & |[e_3]| \cdot |[e_2]| + |[e_3]| \cdot |[e_4]| & \geq & |[e_1]| \cdot |[e_2]| + |[e_3]| \cdot |[e_4]| \\ \Leftrightarrow & |[e_1]| \cdot |[e_4]| + |[e_3]| \cdot |[e_2]| & \geq & 0 \end{array}$$

The claim now follows by induction. □

A bound for Φ_{I^+}

The bound for the inference measure Φ_{I^+} turns out to be similar to the upper bound for the *Rand*-based measure:

$$\Phi^*{}_{I^+}(\mathcal{D}, \mathbb{S}^*, p) = \frac{1}{|\mathcal{D}|} \sum_{[e] \in \mathcal{D}/\sim_{\mathbb{S}^*}} \min\left\{|cov(p, [e])|, |\overline{cov(p, [e])}|\right\}$$

We need to prove that:

Theorem 3.2.3. *Given pattern sets $\mathbb{S}, \mathbb{X}$, pattern p:*

$$\Phi_{I^+}(\mathcal{D}, \mathbb{S}^* \cup \mathbb{X}, p) \leq \Phi_{I^+}(\mathcal{D}, \mathbb{S}^*, p).$$

Proof. Just as before, this claim can be proven using induction: We need to show that

$$\forall p, p_x \in \mathbb{X} : \Phi_{I^+}(\mathcal{D}, \mathbb{S}^*, p) \geq \Phi_{I^+}(\mathcal{D}, \mathbb{S}^* \cup \{p_x\}, p).$$

Again, the measure Φ_{I^+} is defined as a normalised sum. Thus, as for the Rand based measure, it is sufficient to show that each summand is monotonically decreasing. That is, $\forall p, p_x \in \mathbb{X}$ and $\forall [e] \in \mathcal{D}/\sim_{\mathbb{S}^*}$:

$$\min\left\{|cov(p, [e])|, |\overline{cov(p, [e])}|\right\} \geq \sum_{[e'] \in [e]/\sim_{\{p_x\}}} \min\left\{|cov(p, [e'])|, |\overline{cov(p, [e'])}|\right\}.$$

As before let $[e_1], [e_2], [e_3], [e_4] \in [e]/\sim_{\{p_x,p\}}$ be four pairwise disjunct blocks such that $cov(p, [e]) = cov(p, [e_1] \cup [e_3]) = |[e_1] \cup [e_3]|$ and $cov(p_x, [e_1] \cup [e_2]) = |[e_1] \cup [e_2]|$. We can now reformulate the inequality above as:

$$\min(|[e_1] \cup [e_3]|, |[e_2] \cup [e_4]|) \geq \min(|[e_1]|, |[e_2]|) + \min(|[e_3]|, |[e_4]|)$$

Assuming $|[e_1] \cup [e_3]| \leq |[e_2] \cup [e_4]|$ we obtain

$$|[e_1]| + |[e_3]| \geq \min(|[e_1]|, |[e_2]|) + \min(|[e_3]|, |[e_4]|)$$

If $|[e_1]| \leq |[e_2]|$ and $|[e_3]| \leq |[e_4]|$

$$|[e_1]| + |[e_3]| = |[e_1]| + |[e_3]|$$

Otherwise, if $|[e_1]| > |[e_2]|$, $|[e_3]| < |[e_4]|$ (and vice versa) and

$$|[e_1]| + |[e_3]| > |[e_2]| + |[e_3]|$$

The inequality therefore holds and the claim follows through induction. □

3.2.8 Experimental 1: size and partitions of pattern sets, order-restricted hill-climbing

In a first experiment, we considered order-restricted hill-climbing and ask the following questions:

Q3.5 Are small pattern sets mined by this approach?

Q3.6 Is the partition induced by $\mathbb{S}$ recovered well by $\mathbb{S}^*$?

Q3.7 Which effect have the orders on $\mathbb{S}^*$?

Q3.8 Which effect have the different measures on mining?

To evaluate the presented approach we used pattern sets from five UCI itemset mining tasks, harvested using an Apriori implementation (Borgelt 2004) with different minimum support thresholds.

Table 3.7 lists in the first column the support threshold in percent, and in the second the number of closed patterns mined on the data sets. It is noticeable that even though we used relatively high thresholds[3] and the restriction to closed patterns that the result sets are still rather large. Particularly in comparison to the local pattern mining results used in Sections 3.1.1 and 3.1.2, the number of patterns (and therefore the size of the search space) is much larger.

The next three rows report on the size of smallest pattern set that was mined with a Φ on a data set, under any order. For comparison, the minimum for order-independent (upper bound-restricted) search is shown. Underneath, the number of cells into which the full result set partitioned the data is listed, as well as the minimum and maximum number of patterns needed to recover this partition. The minimum number is always the result of $\trianglelefteq_{s^\downarrow}$ or $\trianglelefteq_{l^\uparrow}$, and the maximum number the result of $\trianglelefteq_{s^\uparrow}$ or $\trianglelefteq_{l^\downarrow}$. Using a smaller support threshold (and therefore mining more patterns) roughly equates to adding patterns to the end of the order for $\trianglelefteq_{s^\downarrow}(\trianglelefteq_{l^\uparrow})$ and to the front of it for $\trianglelefteq_{s^\uparrow}(\trianglelefteq_{l^\downarrow})$. Therefore the minimum number of patterns does not change with the support threshold while the maximum number does.

As we can see, the size of the pattern sets is rather small in all cases, far smaller than the, already somewhat reduced, set of closed patterns that was originally mined. So, in answering Question 3.5, we can state that order-restricted

[3] Other work often reports values of only 1%

Dataset	TicTacToe		Mushroom	Breast-Cancer		Vote		Primary-Tumor	
Size	958		8124	286		435		339	
$\theta_{\mathbf{sup}}$	10	5	25	5	1	25	10	25	10
c. patterns	191	951	694	1018	6358	2063	15433	4873	17384
Φ_Q	5	6	3	3	3	5	2	3	3
Φ_I	3	3	3	4	4	3	3	2	6
Φ_C	2	2	2	2	2	2	2	2	2
Φ_{Q^+}	5	4	3	2	2	3	2	3	3
Φ_{I^+}	3	3	3	3	3	3	3	4	4
Φ_{C^+}	1	1	1	1	1	1	1	1	1
max cells	958		1180	266		337	342	258	275
min patterns	17		21	23		26		19	
max patterns	160	78	71	133	231	161	229	88	141

Table 3.7: Data sets and the size of the according *closed* pattern sets, the smallest reduced sets for each measure, and the minimum and maximum number of patterns for the maximal partition as well as its cardinality.

pattern set mining returns compact pattern sets, *if* the right order is used. On the other hand, $\trianglelefteq_{s^\uparrow}$ and $\trianglelefteq_{l^\downarrow}$ lead to larger pattern sets, already giving an indication as to the answer to Question 3.7. The basic constraint, which forms an integral part of the algorithm, also ensures that the partition can be recovered perfectly, yet this is not a satisfying answer to Question 3.6.

To explore this question in more detail, we performed experiments using the three selection measures and four orders described in Sections 3.2.3 and 3.2.4. The measure Φ_C using *clustering* turned out to be the most expensive computationally. The *quotient* measure Φ_Q was rather fast, leaving the measure Φ_I employing a *learning algorithm* in the middle. For each of the three techniques a threshold t has to be supplied which – although indirectly – determines the size of the resulting pattern set $\mathbb{S}^*$. We used the thresholds $\{0, 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4\}$, obtaining one reduced pattern set per threshold for each of the itemsets. The intervals between different thresholds thus become wider the tighter the thresholds are, since we work under the assumption that tightening the thresholds for relaxed values will have a larger effect.

Figure 3.3 shows $\frac{|\mathcal{D}_{\sim^*_{\mathbb{S}}}|}{|\mathcal{D}_{\sim_{\mathbb{S}}}|}$ plotted against $|\mathbb{S}^*|$ for the three measures Φ_Q, Φ_I, and Φ_C, with each curve corresponding to an order. Additionally, a curve $\frac{2^{|\mathbb{S}^*|}}{|\mathcal{D}_{\sim_{\mathbb{S}}}|}$, denoted with *max*, is shown for comparison. This curve represents the "ideal" pattern set, the one that uses the minimum numbers to induce the original partition. Therefore, the closer to *max* a method's plot lies, the closer to the ideal solution it comes.

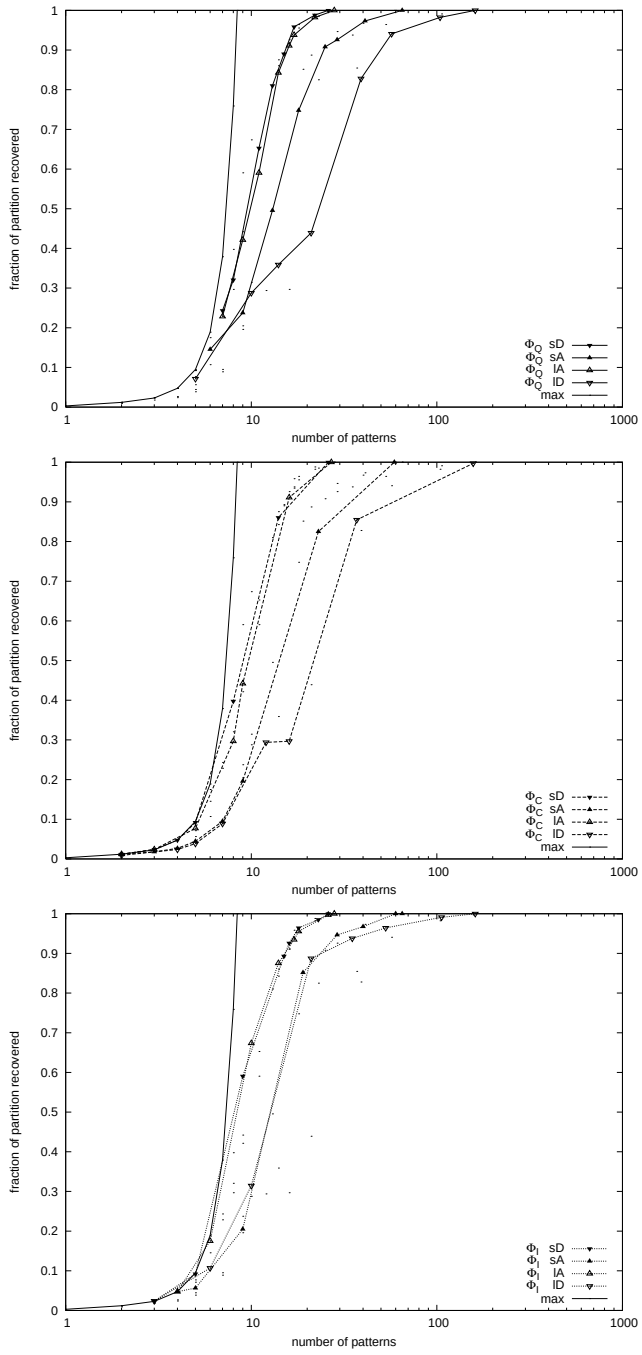


Figure 3.3: Plots for the three measures on Voting-record 25 showing nicely the characteristics present in almost all data sets analyzed. The *max* in each plot indicates the maximum number of cells that can be induced by the given number of patterns.

These particular plots were derived on the *voting-record* set with a minimum support threshold of 25%. The x-axis is scaled logarithmically to make the differences for the smallest pattern sets more visible since many methods converge there. As these plots compare the ability to recover a partition of equal informativeness as the original pattern set traded-off against the size of the reduced set, points/curves closer to the upper left corner can be considered better.

The first observation to be made is that the reduction in patterns does for most settings not lead to a linear decrease in the number of cells, leading to relatively steep curves. This means that our approach does not need to sacrifice too much in information about the data set to improve comprehensibility by the user. Or in other words, even for tightened thresholds (leading to smaller pattern sets), the partition is recovered well, answering Question 3.6.

Second, there are two very distinct differences between Φ_C, and Φ_I and Φ_Q. On the one hand, in the area where the thresholds are rather loose, leading to relatively large $\mathbb{S}^*$, Φ_I and Φ_Q show a smooth curve that corresponds to the small increases in the threshold setting. Φ_C, in contrast, reduces the cardinality of $\mathbb{S}^*$ rapidly. It also induces a coarser (fewer cells but of higher cardinality) partition doing so. On the other hand, once the threshold is raised above a certain value, large changes in the threshold have less effect on Φ_C than on the other two measures. Such changes therefore result in a smooth curve for small sets for Φ_C, which means that although the size of $\mathbb{S}^*$ increases, the number of cells grows slowly, and more abrupt changes for Φ_I and Φ_Q. Ultimately, a threshold of 0.4 for Φ_C leads to a reduction of $|\mathbb{S}^*|$ to 2, essentially the lowest reasonable cardinality, while Φ_I and Φ_Q produce larger sets at that threshold.

Regarding the comparison of Φ_I to Φ_Q, it should be noted that for the "low-support" orderings Φ_I shows a steeper descent for low thresholds which means that it is quicker in recovering the size of the partition.

The behavior for Φ_Q is not surprising since it measures the ability of patterns to split existing cells into sub-cells. It is to be expected that raising the threshold in small steps will exclude only a few patterns compared to the setting before so that smooth curves are produced. In contrast, large steps will ratchet up the selectivity quite a bit, leading to larger decrements in cardinality.

As explained before, Φ_I would for instance reject a pattern that splits only one (or few) "impure" cell(s), especially if they are small since then the pattern could be reliable inferred on the majority of instances. On the other hand, patterns that split large (but few) impure cells, and are therefore accepted by Φ_I, might still be rejected by Φ_Q. This means that especially for relatively low thresholds fewer patterns are removed from consideration, thus allowing to quicker find near-optimal patterns regarding partitioning the data.

Finally, the main operational difference between Φ_C and the other two measures lies (as mentioned in Section 3.2.3) in the fact that the effect of rolling back an "old" decision affects acceptance of a new pattern. This means that two types of patterns will be accepted: those that split many cells (as it should be) and those that evaluate the same for similar pattern combinations. To illustrate this, consider the bit string tree in Figure 3.4: Even though the fourth pattern splits only two cells, the others evaluate similarly given the presence/absence of

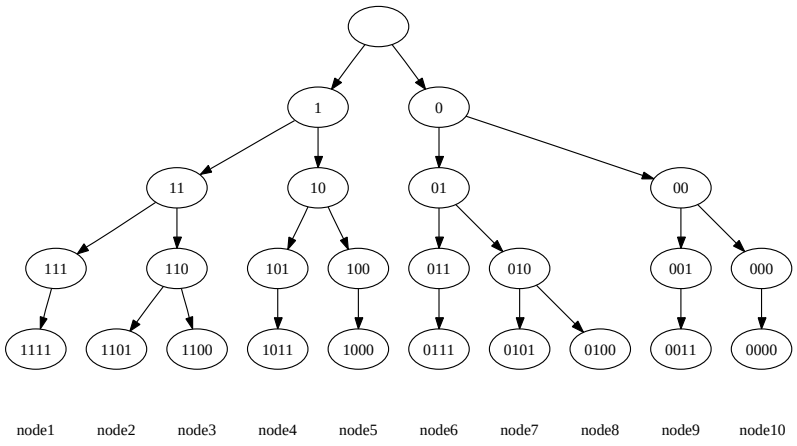


Figure 3.4: Illustrating the clustering criterion

the second and third patterns. *node1* and *node6* are therefore good candidates for merging, as are *node4* and *node9*, or *node5* and *node10*, for instance. Since two merges have to happen to reach the same number of cells as in the level before, the pattern might be accepted, even if the two cells it split are relatively small.

So to summarize:

- Φ_Q will very likely accept.
- Φ_I will accept if unsplit cells are sufficiently impure.
- Φ_C will accept if the pattern is not too *diverse* over the other cells.

In early steps, both impurity and low diversity with regard to prior pattern evaluations are easy to accomplish. While impure cells might still exist in later steps, the bit string formed by prior evaluations quickly takes over so that only patterns are accepted by Φ_C that split *many* cells. This leads to the rather rapid reduction in pattern set cardinality that we observe for Φ_C for lenient thresholds.

For stricter thresholds, only patterns that split large or many cells, are accepted by Φ_I while, as we saw, even patterns that effect insubstantial splits can be accepted by Φ_C if the clustering algorithm merges the right cells. This leads to easier acceptance of patterns at stricter thresholds, leading to the smoother curves (growth in pattern set cardinality accompanied by slower growth in the cardinality of the partition) observed for those threshold values. All of these aspects go into answering Question 3.8.

Third, it can be seen that the "high-support" orderings ($\trianglelefteq_{s^\downarrow}$ and $\trianglelefteq_{l^\uparrow}$) perform very similar to each other – as do the "low-support" orderings ($\trianglelefteq_{s^\uparrow}, \trianglelefteq_{l^\downarrow}$). Additionally, the latter induce larger $\mathbb{S}^*$ for the same number of cells when

compared to the former. Based on these observations we perform the following comparisons to illustrate the relationship between different orders used in hill-climbing.

Comparison of $\trianglelefteq_{s^\downarrow}$ *vs* $\trianglelefteq_{l^\uparrow}$

The similarities of these two orders are not that surprising, given that in pattern mining short (general) patterns usually match relatively many transactions, which means that they have high support. This, however, does only hold to a certain degree[4].

We are interested in how similar the $\mathbb{S}^*$ induced by those two orders are. This is evaluated for each method with regard to the orders in question. We use the *Rand* index as a similarity measure for any two pattern sets, which will decrease if there are a different number of cells or instances are partitioned differently. Since a partition induced by a $\mathbb{S}^*$ constructed using one order will not necessarily have a corresponding partition of equal cardinality, we compare to the two closest partitions (one with higher, one with lower cardinality) and consider the larger *Rand* index.

The partitions induced using the two orders are very similar for all three methods, especially for permissive thresholds reaching *Rand* values in excess of 0.95. The similarity stays high for Φ_I and Φ_Q while Φ_C derives rather different partitions for tight thresholds. So in partially answering Question 3.7, we can state that orders encoding similar phenomena lead to similar pattern sets.

Comparison $\trianglelefteq_{s^\uparrow}(\trianglelefteq_{l^\uparrow})$ *vs* $\trianglelefteq_{s^\downarrow}(\trianglelefteq_{l^\downarrow})$

Obviously, the patterns selected from descending and ascending orders will not be the same. It is, however, possible that patterns selected from one order can be inferred from the other pattern set. To evaluate this, we compare pattern sets resulting from descending vs. ascending orderings while keeping all other variables fixed. Similarly to the *inference selection step*, we induce a model on $\mathcal{D}$ for each pattern of $\mathbb{S}_1^*$ using the patterns of $\mathbb{S}_2^*$ as features, and vice versa. The minimum accuracy for each pattern set is a quantifier of how well one pattern set can be inferred by another one.

Indeed, for $\mathbb{S}^*$ inducing maximal partitions on the data set we observe very high accuracies regarding inference. When the $\mathbb{S}^*$ induce partitions with fewer cells, the inference accuracy decreases as well. However, $\mathbb{S}^*$ obtained using a $\trianglelefteq_{s^\downarrow}$ ordering can usually infer larger sets obtained using a $\trianglelefteq_{s^\uparrow}$ ordering. This observation suggests the following: to recover the complete partition with a compact (understandable) $\mathbb{S}^*$, choose e.g. $\trianglelefteq_{s^\downarrow}$. To discover information not encoded in $\trianglelefteq_{s^\downarrow}$-sets, tighten the thresholds and use $\trianglelefteq_{s^\uparrow}$, since now $\trianglelefteq_{s^\uparrow}$-sets become manageable.

We also compared partitions against each other, using the *Rand* index. While for Φ_I, and Φ_C, usually strong similarities are shown, the situation

[4]Some patterns consisting of 2 items might have higher support than some single item patterns for instance.

changes for Φ_Q. In fact, for data sets where one can truly speak about low-support patterns, such as Breast-Cancer and TicTacToe, *Rand* values drop as low as 0.5, showing that very different partitions are induced. While especially in the first steps, the other measures also choose patterns with low coverage that only split off small cells, this is corrected later. Φ_Q does not include the size of cells in its score and therefore does not recover.

For the re-formulated measures, very similar behavior regarding the minimum size of pattern sets mined can be observed over all data set. The only notable difference, also reported in Table 3.7, lies in the fact that Φ_{C^+} selects only a single pattern for all data sets for the strictest threshold. The chance effect of the agglomerative approach therefore **does** change the outcome of the mining operation. Since different orders do not have an effect on the result set there are no "maximum" number of patterns for the original partition.

This also means that comparison of the partitions using the *Rand* index are only meaningful between different measures used. When those results are compared against each other, the strong similarity is even more pronounced than with order-restricted methods. *Rand* are values usually above or around 0.95, only decreasing for the strictest threshold, 0.4.

3.2.9 Experimental 2: Comparing the prediction quality of selection methods

As we have seen, the size of reduced pattern sets is small enough to allow a human to inspect them. We have argued before, however, that small sets also benefit machine learning techniques that use them as features. Therefore several questions arise:

Q3.9 Are the subsets $\mathbb{S}^*$ better suited as classification features than the full local pattern mining solution $\mathbb{S}$?

Q3.10 Are the smallest $\mathbb{S}^*$ also the best features?

Q3.11 Does upper-bound ordered hill-climbing find better solutions than order-restricted hill-climbing?

To evaluate this we use C4.5 to induce models on the binary feature representation obtained by using the different $\mathbb{S}^*$ and estimate their classification accuracy, via ten-fold cross-validation. We also learn models on the binary vector set constructed using $\mathbb{S}$ and on the original attribute-value representation.

An example for accuracies attainable with different selection measures and orderings is shown in Figure 3.5. The orderings from left to right are $\trianglelefteq_{s^\downarrow}$, $\trianglelefteq_{s^\uparrow}$, $\trianglelefteq_{l^\downarrow}$, $\trianglelefteq_{l^\uparrow}$. The column furthest to the right shows the performance of the pattern set selected by upper-bound ordered hill-climbing, since those are ordering-independent. It can be seen that the "high support" orders are performing better than their respective "low support"-counterparts. While this does not hold for *all* data sets, it is a rather common trend. It is interesting to see that using a selection method with upper-bound pruning is not an effective

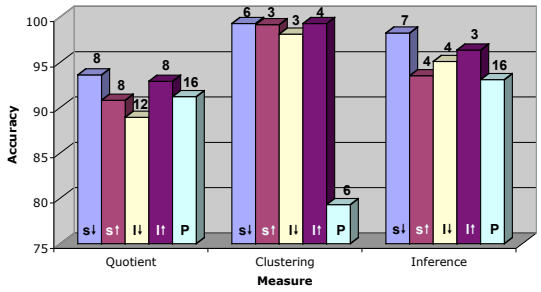


Figure 3.5: Best cross-validated C4.5 accuracies (all orderings for each measure, and the upper-bound approach) on TicTacToe (10%)

strategy for TicTacToe. Selecting the highest-scoring pattern usually means that rather balanced cells are formed, which is detrimental to classification performance on a data set where small subgroups are of relevance. The upper-bound selection performs closest to the order-restricted approach for Φ_Q, the one measure where size of cells (and thus balance) is not considered. The numbers above the columns denote the size of $\mathbb{S}^*$. It can be seen that on the one hand upper-bound ordered search creates larger sets. On the other hand, that the sets giving rise to the best accuracies are typically smaller than the minimum number of patterns needed for the original partition.

We use a second figure (Figure 3.7) for comparing the accuracies of C4.5 models on five representations. These are

- the original attribute-value representation of the data, and
- binary vectors which are created using $\mathbb{S}$,
- the best-performing $\mathbb{S}^*$ selected by an order-restricted approach,
- $\mathbb{S}^*$ returned by the upper-bound ordered selection step, and
- *Pattern teams* returned by the *maximum informative itemsets* method (*miki*s), introduced by Knobbe *et al.* (2006a), respectively.

At the top of the bar representing the best $\mathbb{S}^*$, a number denotes the size of the corresponding pattern set. The most relevant result of this comparison is that usually neither the binary vector representation derived from the whole $\mathbb{S}$, nor the one based on the $\mathbb{S}^*$ that gives the maximal partition are best-suited for the machine learning algorithm. Instead, for all data sets, a reduced pattern set gives the best accuracy after cross-validation and pruning. This supports our assumption that too many features only lead to an over-fitting effect and do not benefit the learner. The size of the corresponding pattern sets is so small that they are easily interpretable by the user. An example for the *tic-tac-toe* data set is given in Figure 3.6. What can also be observed is that the weak performance

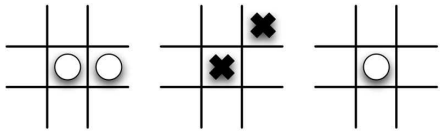


Figure 3.6: A visualisation of $\mathbb{S}^*$ for Tic-Tac-Toe 5% using $\trianglelefteq_{s^\downarrow}$, Φ_C, and a threshold of 0.4.

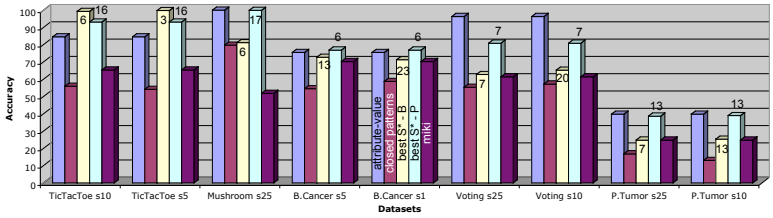


Figure 3.7: C4.5 cross-validated accuracies for attribute-value representation, closed-set binary representation, best $\mathbb{S}^*$, best $\mathbb{S}^*$using upper-bound pruning, and *miki* (2 patterns)

of the upper-bound heuristic mining on TicTacToe was indeed an exception. On the other data sets, it achieves the best accuracy of all the pattern sets, while at the same time still having relatively low cardinality.

The attribute-value representation still proves to be more expressive than the pattern representations for most cases. Such a straight-forward representation is not possible for structured data, however, and as we have shown in a different work (Bringmann et al. 2006), pattern subset selection is very important for structure-activity prediction, for instance. Furthermore, even though the order-restricted hill-climbing always performs well, no single order or selection method does distinguish itself and selection thresholds vary between 0.03 and 0.4. This is particularly pronounced in the case of TicTacToe. Keep in mind however that we didn't aim to maximize predictive accuracy and our illustrative instantiations are not meant to exhaust the entire issue.

To summarize:

- Answering Question 3.9: A mined subset $\mathbb{S}^*$ is preferable to the full set $\mathbb{S}$.
- Answering Question 3.10: While small sets perform well, the best accuracies are not achieved by the smallest sets.
- Answering Question 3.11: Upper-bound ordered mining finds better patterns for classification purposes, due to its more flexible mining scheme.

3.2.10 Experimental 3: Comparison to pattern teams

An alternative we mentioned before lies in restricting the size of pattern sets and searching for pattern sets that maximize *joint entropy* exhaustively. This technique was introduced by Knobbe *et al.* (2006b). We aim to explore

Q3.12 How similar are the solutions of the exhaustive and the heuristic methods?

Due to the rather large running times of these techniques only pattern teams of size $k = 2$ have been selected. The comparisons in this case focus on the similarity of the pattern teams to our reduced pattern sets (*Rand*-index and inference) and on the effectiveness as features for classification purposes.

We compared the maximally informative k-itemsets (*miki*) obtained with algorithms by Knobbe *et al.* to reduced sets obtained by using Φ_C with a rather strict threshold. We chose Φ_C since it mines the smallest pattern sets. Interestingly the sets compared exhibit very similar behavior with regard to all comparisons. First, all but one consist of two patterns. Second, their prediction qualities are the same, and third they can both equally well infer the other set. High *Rand* values furthermore show that our approach induces very similar partitions. This is interesting insofar as *miki*s are mined using a complete method maximizing joint entropy, a measure that rewards balanced partitions, while we employ a heuristic method. All in all, the heuristic and exhaustive methods lead to very similar solutions, at least for small pattern set cardinality. The exhaustive technique has the advantage that it can constrain this cardinality directly, while our approach can be expected to be more efficient.

3.2.11 Conclusions

We discussed the related work that exists in terms of heuristic pattern set post-processing in Section 2.2.4. Given the results of our experimental evaluation, two main consequences can be drawn.

First, we showed that upper-bound ordered hill-climbing leads to high-quality sets that are unaffected by any order used for mining. At the same time, the upper bound allows to retain mining efficiency, an important characteristic, since the use of fixed orders stems mainly from trying to control computational complexity. Formulating the right measures to allow us to calculate upper bounds is not trivial. Defining such measures would allow to make techniques such as CBA, CMAR, Krimp more independent from heuristic parameters, as Picker* is more independent from them than Bouncer. Even if the formulation of upper-boundable measures is not possible, additional thought should be given to the use of different orders from the ones usually employed in order-restricted systems. The orders used in CBA, CMAR, or Krimp are well-motivated but the authors typically acknowledge that there might be other ones that could be more useful. Analyzing the desired properties of pattern sets further could lead to the development of alternative orders.

Second, we compared our technique to an exhaustive method optimizing joint entropy (Knobbe and Ho 2006a) and showed that the resulting pattern

sets are of very similar quality. This means that the size-restriction that is used to limit the cost of pattern set mining can be abandoned without losing much in terms of quality and efficiency.

3.3 Summary

The first type of techniques which we discussed in this work is concerned with mining unordered pattern sets in a post-processing step. We contrasted two approaches addressing this task in this chapter:

1. Exhaustive mining which can guarantee optimal results and gives the user much control over the mining process via constraints.

2. Heuristic mining which will in all likelihood be much faster.

Both of these approaches are newly developed by applying existing techniques from data mining and machine learning, respectively, to the pattern set mining task. We have published these techniques and the experimental results before as (De Raedt and Zimmermann 2007) and (Bringmann and Zimmermann 2009). Exhaustive mining trades off the optimality of the solution against the problems that stem from deriving large parts of a search space that spans the power set of a high amount of local patterns. As we showed, solutions satisfying the constraints set by a user can be found relatively efficiently – but this depends strongly on the constraint parameters chosen. In this way the exhaustive technique mirrors the local pattern mining techniques on which it is based. Additionally, unless restricting the solution by a top-k constraint, the number of returned pattern sets can be rather large.

As an alternative, we explored heuristic approaches in the second section, which do not explicitly reduce redundancy between patterns but instead evaluates a pattern set in terms of the partition it induces. Evaluating both an order-restricted and an upper-bound ordered hill-climbing technique, we focussed on gaining an understanding of the effects that orders and different measures have on the resulting sets. A resulting pattern set can of course not be guaranteed to be optimal but as the experimental evaluation showed, these techniques mine useful pattern sets efficiently.

While unordered pattern sets are very attractive due to their flexibility, it can be more effective in certain contexts to use ordered pattern sets. Especially in the case of order-restricted pattern set mining, the resulting pattern set intuitively can be considered an ordered set. The order on such sets can be imposed during mining itself or after the fact. We will explore those alternatives and the effects they have on the quality and composition of the pattern sets in the next chapter.

Chapter 4

Evaluating the Effects of Orders in Ordered Sets

The techniques discussed in the preceding chapter are mainly used to mine unordered sets. Unordered sets have the advantage that they can be employed rather flexibly, for descriptive purposes as well as as features for classification tasks. On the other hand, it might be desirable to use patterns directly for classification or to get a sense of which patterns are more important than others. For such purposes, ordered sets are more useful.

An order could be imposed on an unordered pattern set afterwards but may be somewhat arbitrary in this case, and therefore hard to justify. On the other hand, orders can arise from the local pattern mining mining operation, or from the pattern set mining operation itself.

In the first section of this chapter, we contrast the effect of imposing an order by post-processing or having it emerge during mining. Specifically, exhaustive search for patterns that satisfy certain constraints produces unordered sets which can be ordered by defining an order on the existing set. Top-k mining creates this order during the local pattern mining step itself, as part of enforcing the top-k constraint. Which of those two approaches is chosen has an effect both on the size of pattern sets and on their effectiveness when it comes to classification. The order imposed by top-k mining also lends itself to a straight-forward but effective post-processing step which reduces the cardinality of pattern sets even further while *improving* accuracy. This pattern set selection step uses a validation set to remove higher-ordered patterns, that is, patterns with lower scores.

A more informed post-processing method is order-restricted hill-climbing. As the name already implies, the order in which patterns are processed is very important to this technique. The second section therefore addresses the effect that different orders have on this approach. We again contrast an arbitrary order imposed after local pattern mining with the order that arises from top-k mining and show how the orders affect the results of pattern set mining. There

are pronounced differences between the cardinality of pattern sets as well as with regard to the classification accuracy of classifiers based on these pattern sets. Additionally, our experiments provide some evidence regarding the effect on optimality of using heuristic post-processing.

4.1 CTC - Order by top-k mining

The contents of this section, published in (Zimmermann and Bringmann 2005), are the result of joint work with Björn Bringmann at the Katholieke Universiteit Leuven.

As we mentioned above, top-k mining induces an intuitive order on the set of patterns returned:

$$\forall p_1, p_2 \in \mathcal{L}_p : (p_1 \lhd p_2) \Leftrightarrow \sigma(sp(p_1)) > \sigma(sp(p_2)) \tag{4.1.1}$$

and therefore is a straight-forward way of mining ordered pattern sets $(\mathbb{S}, \trianglelefteq)$.

The alternative lies in imposing an order on the patterns *after* they have been mined, for instance, based on their support and confidence. Especially when using predictive patterns as a decision list afterwards, the order can have a pronounced effect on the quality of the resulting classifier. To evaluate the usefulness of the top-k constraint, we evaluate the performance of such simple ordered pattern sets in the context of classification against a classifier constructed from an unordered result set of local pattern mining. We would expect the more intuitive order arising from top-k mining to lead to effective classifiers. Ordered pattern sets also lend themselves to a variety of post-processing strategies, one of which, order-restricted hill-climbing, we discussed in the preceding chapter. In this section, we use a simpler technique, which removes higher-ranked patterns (those with lower score) based on the performance of the classifier on a validation set. In addition to the evaluation of the effectiveness of classifiers based on these sets, we evaluate the cardinality of the pattern sets deriving from the two different mining approaches. Since top-k mining enforces an implicit pattern set constraint that can be expected to have an effect on how patterns relate to each other (and their redundancy), we expect to see this reflected in the result set.

Both the pattern language and the data language are that of rooted trees as defined in Definition 1.1.4, and the matching function *tree embedding* as defined in Definition 1.1.7. We use this particular matching function to compare our approach (CTC for **c**orrelated **t**ree patterns for **c**lassification) with Zaki *et al.*'s technique XRULES (2003). This notion is more flexible than simple subtrees and the mining process is still efficient.

The local pattern mining operation performed in Zaki *et al.*'s work is exhaustive constraint-satisfaction mining. Each instance in the data set is annotated one of two class label $C = \{c_1, c_2\}$. Target patterns therefore take the form $C = c_i$, and patterns mined by Zaki *et al.*'s XRULES technique have to satisfy

the constraints:

$$Th_{XRules}(\mathcal{L}_{tree}, \mathcal{D}, c) = \{p \in \mathcal{L}_{tree} \mid sup(p) \geq 0.01 \wedge (\exists c_i \in C : conf(p \Rightarrow c_i) \geq 0.5)\} \tag{4.1.2}$$

We contrast this technique with a solution of our own, called CTC (**c**orrelating **t**ree patterns for **c**lassification), which performs top-k mining, using $\sigma = \chi^2$ and an additional minimum significance constraint:

$$Th_{CtC}(\mathcal{L}_{tree}, \mathcal{D}, c) = \{p \in \mathcal{L}_{tree} \mid \chi^2(sp(p)) \geq \theta_{sig} \wedge p \in \arg_k \max \chi^2(sp(p))\}$$

We extend the order mentioned in Equation 4.1.1 in the following way:

$$\begin{aligned} \forall p_1, p_2 \in \mathcal{L}_p : (p_1 \lhd_{CtC} p_2) \text{ if and only if } \sigma(sp(p_1)) > \sigma(sp(p_2)) \text{ or} \\ \sigma(sp(p_1)) = \sigma(sp(p_2)) \wedge |V_{p_1}| < |V_{p_2}| \end{aligned} \tag{4.1.3}$$

V_{p_1} and V_{p_2} are the node sets of the respective trees p_1, p_2, as defined in Definition 1.1.4. This means that in case that two patterns achieve the same score, we will consider the one that involves fewer nodes to come first. Zaki *et al.* also mention an order based defined by Liu *et al.* (1998): Given two patterns $p_1 \Rightarrow c, p_2 \Rightarrow c$, $p_1 \Rightarrow c \trianglelefteq_{XRules} p_2 \Rightarrow c$ if and only if:

$$\begin{aligned} &conf(p_1 \Rightarrow c) > conf(p_2 \Rightarrow c) \text{ or} \\ &conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) > sup(p_2 \Rightarrow c) \text{ or} \\ &conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) = sup(p_2 \Rightarrow c) \wedge |V_{p_1}| < |V_{p_2}| \text{ or} \\ &conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) = sup(p_2 \Rightarrow c) \wedge |V_{p_1}| = |V_{p_2}| \wedge \\ &p_1 \text{ occurs lexicographically before } p_2 \end{aligned} \tag{4.1.4}$$

The goal in concept learning, as we explained in Section 1.2.1, is to approximate a function $\mathcal{L}_p \mapsto C$ which accurately predicts the class labels of unseen instances. In pattern-based classifiers, usually a combination of the mined patterns and a strategy for aggregating each pattern's prediction is used. Given an unseen example $e_?$, Zaki *et al.* define the *average strength* (AvgStr) strategy, which takes the form:

$$\begin{aligned} &f_{AvgStr}(e_?) = \\ &\arg_{c \in C} \max \frac{1}{|\{p \in Th_{XRules} | match(p, e_?) = 1\}|} \sum_{p | match(p, e_?) = 1} conf(p \Rightarrow c) \end{aligned} \tag{4.1.5}$$

An ordered set allows us to use a different classification strategy, that of a *decision list* (DL):

$$f_{DL}(e_?) = \arg_{c \in C} \min_{\trianglelefteq_{CtC}} \{p \in Th_{CtC} \mid match(p, e_?) = 1 \wedge conf(p \Rightarrow c) \geq 0.5\} \tag{4.1.6}$$

The work introducing the XRULES system reports that the *average strength* technique outperforms the *decision list* method. This is an indication that the order $\trianglelefteq_{XRules}$ imposed on the pattern set after mining is not well-suited to

classification. An ordered set can of course also be used as an unordered set and *majority voting* (MV) used:

$$f_{MV}(e_?) = \arg_{c \in C} \max \sum_{p \in Th_{XRules} | conf(p \Rightarrow c) \geq 0.5} match(p, e_?) \tag{4.1.7}$$

It seems somewhat counterintuitive that patterns with different confidence of their prediction (or statistical significance) should count equally strong in an actual voting. A solution to this problem are *weighted voting* strategies, such as the AvgStr technique described above. An alternative is the *weighted*χ^2 heuristic (WChi), introduced by Han *et al.* (2001). It discounts the actual χ^2 value of a rule against the maximum value it could have achieved to calculate a voting weight. In this context we first have to define the $\max \chi^2$ of a rule $p \Rightarrow c$:

$$\begin{aligned} \max \chi^2(p \Rightarrow c) &= \left(\min\{sup(p), sup(c)\} - \frac{sup(p)sup(c)}{|\mathcal{D}|} \right)^2 |\mathcal{D}| e \text{ where} \\ e &= \frac{1}{sup(p)sup(c)} + \frac{1}{sup(p)(|\mathcal{D}| - sup(c))} \\ &+ \frac{1}{(|\mathcal{D}| - sup(p))sup(c)} + \frac{1}{(|\mathcal{D}| - sup(p))(|\mathcal{D}| - sup(c))} \end{aligned}$$

The full classification strategy takes the form:

$$f_{WChi}(e_?) = \arg_{c \in C} \max \sum_{p \in Th_{XRules} | match(p, e_?) = 1} \frac{\chi^2(sp(p))\chi^2(sp(p))}{\max \chi^2(p \Rightarrow c)} \tag{4.1.8}$$

4.1.1 Experimental evaluation

Compared to exhaustive constraint satisfaction local pattern mining, top-k mining considers relations between patterns. Additionally, it induces an ordered set, which differs from the unordered set which the XRules mines.

The questions to be addressed thus are:

Q4.1 Does top-k mining lead to the induction of smaller pattern sets than minimum support, minimum confidence mining?

Q4.2 Are ordered pattern sets better suited for classification than the unordered sets of individually constrained patterns?

The XML data used in our experiments are log files from web-site visitors' sessions. They are separated into three weeks (CSLOG1, CSLOG2, and CSLOG3) and each session is classified based on whether the visitor came either from an `.edu` domain or from any other domain. Characteristics of the datasets are shown in Table 4.1.

In each setting we used one set of data for training and another one for testing. Following Zaki's notation, CSLOGx-y denotes that we trained on set x and tested on set y.

Table 4.1: Characteristics of Datasets (taken from original publication)

DB	#Sessions	edu	other	%edu	%other
CSLOG1	8074	1962	6112	24.3	75.7
CSLOG2	7409	1687	5722	22.8	77.2
CSLOG12	13934	2969	10965	21.3	78.7
CSLOG3	7628	1798	5830	23.6	76.4

Table 4.2: Size of the induced pattern sets for CTC, XRULES

Setting	CTC	XRULES	CTC_{Val}
CSLOG1-2	592	28911	130
CSLOG2-3	497	19098	150
CSLOG12-3	981	29098	170
CSLOG3-1	546	31661	220

For the experimental evaluation, we compared our approach to XRULES on the XML data used in Zaki *et al.*'s (2003) publication. We used the thresholds $\theta_{sup} = 0.3\%$ for CSLOG1-2 and CSLOG2-3, $\theta_{sup} = 0.35\%$ for CSLOG12-3, and $\theta_{sup} = 0.2\%$ as in the original publication, as well as $\theta_{conf} = 50\%$. For CTC, we mined with $k = 1000$ and $\theta_{sig} = 3.84$, the 95% p-value for the χ^2 distribution.

Table 4.2 shows the complexity of the resulting pattern sets. The first column lists the setting for which the corresponding pattern set was induced. The second column reports the number of rules mined by CTC for the parameter setting given above, column three shows the number of rules for XRULES. It is interesting to note that for all four settings the dataset supports less than 1000 rules that pass the 90% significance test. As can be seen, XRULES produces rule sets that are two orders of magnitude larger than those induced by CTC. We can answer Question 4.1 with a clear "yes": the relation of patterns to each other as well as the use of a significance measure therefore leads to far smaller pattern sets.

To explore the effect that varying the number of rules used for classification has on predictive accuracy, we evaluated subsets of the rule sets induced by CTC on the whole test sets. For those subsets the l first rules according to $\trianglelefteq_{CtC}$ induced for a particular setting were selected. The smallest subset had size 10 and we increased l in increments of 10 up to the total number of rules induced for the respective setting. Figures 4.1-4.4 show the resulting error rates for the four different settings. In each diagram, **MV** denotes the majority voting strategy, **DL** the decision list approach, **AvgStr** Zaki *et al.*'s average strength heuristic was used, and **WChi** the use of the weighted χ^2 heuristic introduced by Han *et al.*. It is noticeable that using less than 100 rules causes relatively high error rates while significantly enlarging the rule set past 200 rules causes error rates to increase again. This means that rules that appear later in the order are very likely the result of overfitting.

By using validation sets one can determine the size of the rule set giving best

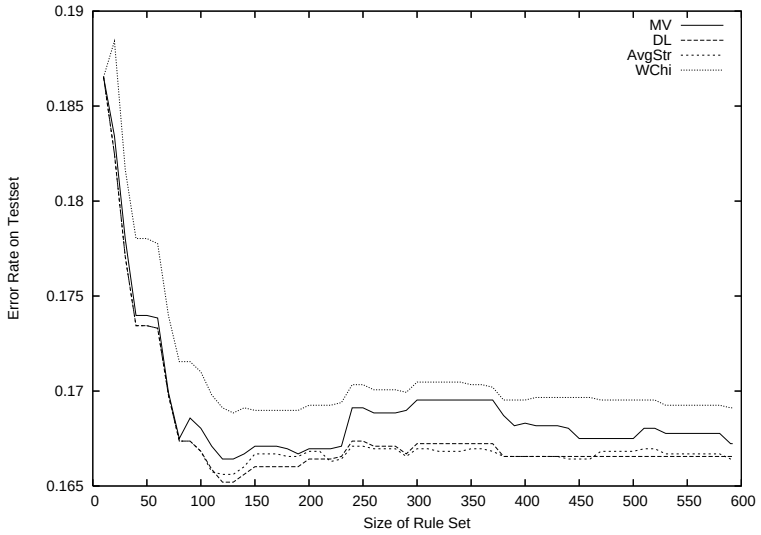


Figure 4.1: Error rates for different classification strategies for the CSLOG1-2 setting

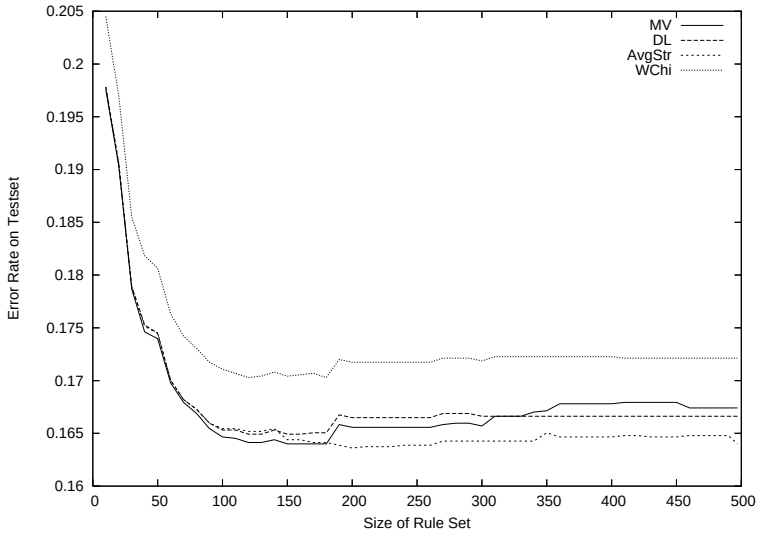


Figure 4.2: Error rates for different classification strategies for the CSLOG2-3 setting

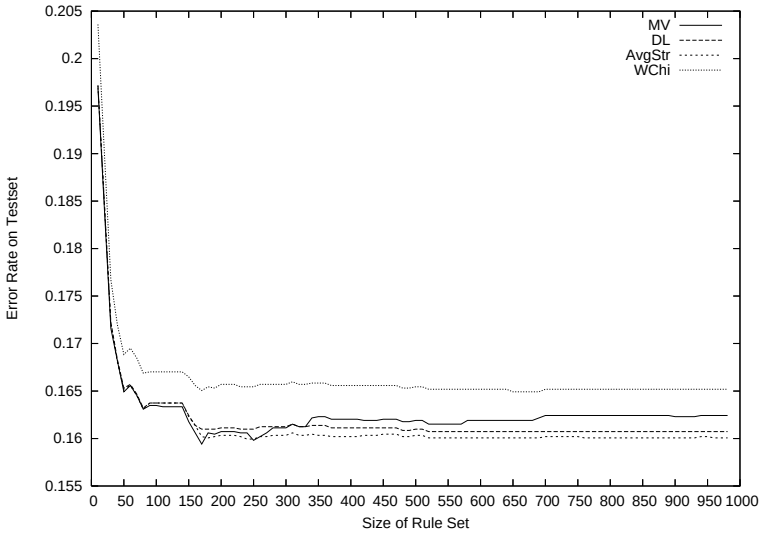


Figure 4.3: Error rates for different classification strategies for the CSLOG12-3 setting

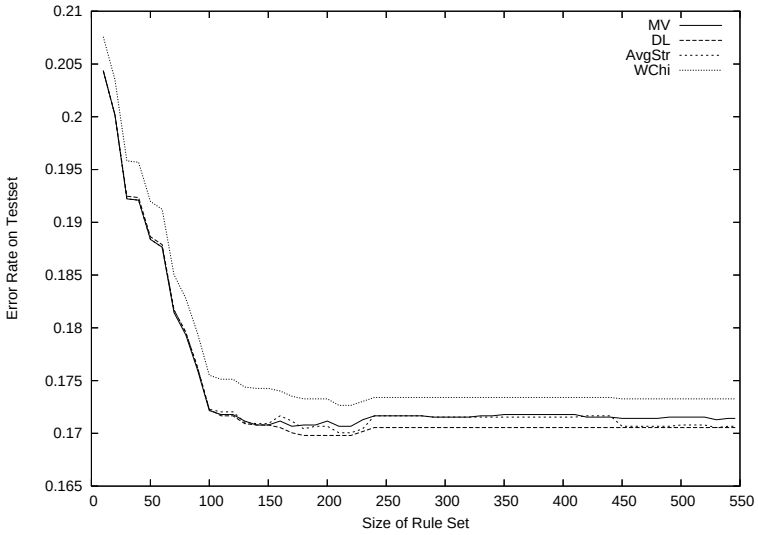


Figure 4.4: Error rates for different classification strategies for the CSLOG3-1 setting

Table 4.3: Predictive Accuracy for XRules, different classification strategies for CtC

Setting	XRules	CtC$_{MV}$	CtC$_{DL}$	CtC$_{AvgStr}$	CtC$_{WChi}$
CSLOG1-2	82.99	83.23	83.31	83.31	83.01
CSLOG2-3	84.61	83.95	83.90	83.92	82.83
CSLOG12-3	85.30	84.27	84.24	84.29	83.53
CSLOG3-1	83.81	83.50	83.77	83.63	83.53

performance. We used half of the corresponding test sets as validation sets. The resulting (best) number of rules for each setting is reported in the last column of Table 4.2, denoted by CtC$_{Val}$. This rather ad-hoc post-processing step proves to be surprisingly effective, however, strongly reducing the size of the pattern set and improving its effectiveness. By using the respective number of rules for each setting and classifying the other half of the test set we arrive at predictive accuracy estimates that we report in Table 4.3.

As can be seen, CtC performs well in all settings, even though XRules always exhibits the highest accuracy. For the first setting, CSLOG1-2, the differences between the different classifiers (XRules, and the four variants of CtC are not significant at the 5% level, as measured by a paired t-test. For the settings CSLOG2-3 and CSLOG12-3, XRules and the first three CtC variants (MV, DL, AvgStr) again show no significant difference. CtC with the weighted χ^2 heuristic performs significantly worse than XRules. Finally for the last setting, CSLOG3-1, the situation is the same as for the CSLOG1-2 setting.

This means that the ordered rule set with an ad-hoc pattern selection step based on the order over the patterns easily captures the same amount of information as the full result of a local pattern mining operation, answering Question 4.2. Additionally, the ordered set allows the use of a simpler classification strategy (the decision list) without losing predictive accuracy.

4.2 CBA and CBC - Order-restricted pattern set mining

As the preceding section showed, ordered sets can be rather effective for classification, allowing to replace sophisticated classification schemes with simpler ones. We also saw that orders arising from the mining process itself can be better suited to this task than orders that are imposed retroactively. Additionally, we augmented the top-k mining with an order-based post-processing step that simply kept the first patterns in the order, evaluating the accuracy on a validation set.

A similar order, which we introduce below, can be used for more elaborate pattern set mining algorithms, however, such as order-restricted hill-climbing. We already encountered this technique when mining unordered pattern set in Section 3.2. In this section, we therefore combine exhaustive constraint satis-

faction and exhaustive top-k mining with order-restricted hill-climbing.

Given the effectiveness of the implicit pattern set constraint in top-k mining, the question is whether this effectiveness in reducing the size of pattern sets is retained when the post-processing step becomes more elaborate. Related to this is the question of whether enforcing the pattern set constraint during local pattern mining will make the entire mining process (local pattern and pattern set mining combined) computationally more expensive. Finally, there is of course the question whether the choice of the local pattern mining step has an effect on the suitability of pattern sets for classification. As an aside, we will investigate whether the heuristic post-processing step is effective in avoiding local minima to construct a well-performing classifier.

Once again, we perform experiments in the context of concept learning. We use itemsets both as the pattern and the data language, once again, with each instance labeled with one of two class labels $C = \{c_1, c_2\}$.

An existing technique that performs exhaustive constraint satisfaction search as a first phase, and order-restricted hill-climbing for pattern set mining, is CBA (Liu et al. 1998). The local pattern mining constraint is the same as for the XRULES system of the preceding section:

$$Th_{CBA}(\mathcal{L}_\mathcal{I}, \mathcal{D}, c) = \{p \in \mathcal{L}_\mathcal{I} \mid sup(p) \geq 0.01 \wedge (\exists c_i \in C : conf(p \Rightarrow c_i) \geq 0.5)\} \tag{4.2.1}$$

On the initially unordered set $Th_{CBA}(\mathcal{L}_\mathcal{I}, \mathcal{D}, c)$, the order $\trianglelefteq_{CBA}$, as defined in Definition 1.3.1 is imposed, which we repeat here: Given two patterns $p_1 \Rightarrow c, p_2 \Rightarrow c$, $p_1 \Rightarrow c \trianglelefteq_{CBA} p_2 \Rightarrow c$ if and only if:

$$\begin{aligned}
&conf(p_1 \Rightarrow c) > conf(p_2 \Rightarrow c) \text{ or} \\
&conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) > sup(p_2 \Rightarrow c) \text{ or} \\
&conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) = sup(p_2 \Rightarrow c) \wedge |p_1| < |p_2| \text{ or} \\
&conf(p_1 \Rightarrow c) = conf(p_2 \Rightarrow c) \wedge sup(p_1 \Rightarrow c) = sup(p_2 \Rightarrow c) \wedge |p_1| = |p_2| \wedge \\
&p_1 \text{ occurs lexicographically before } p_2
\end{aligned} \tag{4.2.2}$$

This order is then used in order-restricted hill-climbing, using the pattern set constraint $\mathcal{C}_{CBA}$, defined in Example 1.3.4:

$$\mathcal{C}_{CBA}(\mathbb{S}, \mathcal{D}) \equiv \forall p \Rightarrow q \in \mathbb{S}, \exists e \in \mathcal{D} : e \notin \bigcup_{p_i \trianglelefteq_{CBA} p} cov(p_i, \mathcal{D}) \wedge class(e) = q$$

Our own system, which combines top-k (using $\sigma = \chi^2$) mining with order-restricted hill-climbing, is denoted by CBC. The order on the pattern set, $\trianglelefteq_{CBC}$

takes the form: Given two patterns p_1, p_2, $p_1 \lhd_{CBC} p_2$ if and only if:

$$\max_{c \in C} conf(p_1 \Rightarrow c) > \max_{c' \in C} conf(p_2 \Rightarrow c') \text{ or}$$
$$\max_{c \in C} conf(p_1 \Rightarrow c) = \max_{c' \in C} conf(p_2 \Rightarrow c') \wedge \sigma(sp(p_1)) > \sigma(sp(p_2)) \text{ or}$$
$$\max_{c \in C} conf(p_1 \Rightarrow c) = \max_{c' \in C} conf(p_2 \Rightarrow c') \wedge \sigma(sp(p_1)) = \sigma(sp(p_2)) \wedge |p_1| < |p_2|$$
$$\max_{c \in C} conf(p_1 \Rightarrow c) = \max_{c' \in C} conf(p_2 \Rightarrow c') \wedge \sigma(sp(p_1)) = \sigma(sp(p_2)) \wedge |p_1| = |p_2| \wedge$$
p_1 occurs lexicographically before p_2 (4.2.3)

4.2.1 Experimental evaluation

After we already established in the comparison between XRULES and CTC that the top-k constraint can be leveraged for mining smaller and more effective pattern sets, the questions for this evaluation are:

Q4.3 Does the order inherent to top-k mining, $\trianglelefteq_{CBC}$ lead to a better pattern set for classification selected than the subsequently imposed $\trianglelefteq_{CBA}$?

Q4.4 Have the different orders on pattern sets an effect on which patterns are selected for patterns sets and on their size?

Q4.5 Does using the pattern set constraint in top-k mining lead to a more efficient mining operation than minimum support, minimum confidence mining?

We used the WEKA-implementation of APRIORI, with $\theta_s = 0.01$ and $\theta_{conf} = 0.5$ for CBA's first phase. For CBC, $\theta_{sig} = 3.84$. In both cases, the number of rules mined was set to $k = 50000$.[1] As before, CBC only mines free patterns. Both classifiers use the decision list strategy for classification, using the final pattern sets.

We chose 13 data sets from the UCI Machine Learning repository (Blake and Merz 1998), and discretized numerical values, using Irani & Fayyad's MDL-based discretization (Fayyad and Irani 1993). Data is represented in $\mathcal{L}_{\mathcal{I}}$ for CBA, an algorithm that is based on itemset mining and therefore also uses this pattern language. For CBC, data stays in its original $\mathcal{L}_{\mathcal{A}}$ format and we use $\mathcal{L}_{AV}$ to represent patterns. To evaluate the quality of built classifiers, a stratified ten-fold cross-validation was performed, and we report on the average classification accuracy and its standard deviation per data set. So as to not have differences of the implementation influence the efficiency estimates, we do not report on running times but instead on the number of candidate patterns evaluated, as a measure for the effectiveness of pruning performed.

Results The accuracy comparison of CBC with CBA (Table 4.4) shows that a certain advantage of the pattern set constraint inherent in top-k mining (and

[1] An exception is the *Kr-vs-KP* data set where 90,000 rules are needed for CBA to find rules with confidence of at least 90%

Table 4.4: **Average accuracy and standard deviation for** CBA **and** CBC**.** The left-most column lists data sets, columns 2 and 3 accuracy estimates based on a stratified ten-fold cross-validation

Dataset	CBA	CBC
Balance (2 Class)	79.18 ± 4.59	79.18 ± 4.59
Breast-Cancer	68.19 ± 8.48	66.77 ± 9.28
Breast-W	94.71 ± 1.9	95.71 ± 1.34
Colic	81.27 ± 8.07	76.91 ± 6.51
Credit-A	85.65 ± 4.35	84.06 ± 4.48
Credit-G	71.4 ± 2.63	69.8 ± 4.89
Diabetes	75.92 ± 4.14	75.78 ± 4.23
Heart-H	83.33 ± 6.69	82.66 ± 5.82
Kr-vs-Kp	80.72 ± 1.75	$95.63 \pm 1.29\circ$
Mushroom	99.53 ± 0.19	$100\circ$
Spambase	86.39 ± 1.62	86.09 ± 1.61
Tic-Tac-Toe	100	100
Voting Record	94.25 ± 3.1	93.1 ± 3.58

$\circ$ denotes statistical wins at the 99% level according to paired t-test

the order it induces) still exists. CBC never performs significantly worse than CBA and in two cases is significantly better, as measured by a paired t-test. Order-restricted hill-climbing does not seem to lead to relevant qualitative differences between the two techniques. In the *Kr-vs-Kp* scenario, limiting the mining process to the $90,000$ most significant rules in CBA excludes many high-confidence rules. Even at $200,000$, the highest confidence is at just 0.92, while for CBC rules with confidence 1.0 are found within the $50,000$ most significant rules according to χ^2. We therefore answer Question 4.3 to the effect that while the top-k constraint somewhat improves the quality of the classifier, the difference is not very pronounced. This is an indicator that the pattern set mining technique recovers potential weaknesses of the choice for local pattern mining.

In the case of the *Mushroom* data set, the ordering of the rule set before pruning is decidedly different between the two approaches and thus different rules are selected for the final classifier. To give an impression of the effect of the different orders used in the pattern set mining step (which arise from the different local pattern mining approaches), consider Table 4.5. It demonstrates nicely what a big impact the order can have in the case of order-restricted hill-climbing. Particularly when one considers that the differences in accuracy (derived using the same classification scheme in both cases) are mostly not significant. We can therefore state with regard to Question 4.4, that the different orders have a noticeable effect on the results of order-restricted hill-climbing as a pattern set mining technique, similarly to the results of Section 3.2.

Notice especially that the small number of patterns that can be achieved by

Table 4.5: **Cardinality of pattern sets after the pattern set mining step**

Data set	CBA	CBC
balance	17.40 ± 5.10	8.00 ± 0.00
breast cancer	98.50 ± 4.74	10.60 ± 1.35
breast	51.40 ± 3.59	12.90 ± 0.57
colic	88.30 ± 7.79	3.00 ± 0.00
credit a	125.00 ± 12.05	2.60 ± 0.69
credit g	57.10 ± 10.71	18.80 ± 3.74
diabetes	59.10 ± 17.82	6.80 ± 1.03
heart	43.20 ± 2.25	8.30 ± 1.42
kr vs kp	32.9 ± 1.45	2.20 ± 0.42
mushroom	26.00 ± 0.00	8.00 ± 0.00
spambase	2.50 ± 0.97	18.90 ± 1.19
tic tac toe	8.00 ± 0.00	10.00 ± 0.00
vote	38.40 ± 2.95	8.00 ± 0.82

exhaustive mining on the Balance data set, as shown in Section 3.1.2, 4.9 ± 1.66 patterns, is not reached by either ordering. The classification accuracy, on the other hand, is the same for all three solutions.

The third question to be answered is whether exhaustive constraint satisfaction mining is more efficient than top-k mining. Table 4.6 shows no clear-cut advantage for either technique. On average CBA mines slightly fewer patterns than CBC, however. This means that the top-k constraint does not lead to a reduction in computational complexity, answering Question 4.5 negatively.

Large data sets on which accurate rules have small coverage, and data sets with minority classes make upper-bound pruning less effective. More specifically, top-k mining compares worst on *Kr-vs-Kp*, *Spambase*, and *Tic-Tac-Toe*. We have seen, however, that *Kr-vs-Kp* gives also CBA trouble and subsequent experiments in which the number of mined patterns is set to 1.8 million still does not result in classifiers comparing well with CBC while exceeding its number of evaluated candidate patterns significantly.

4.2.2 Conclusions

Order-restricted hill-climbing is a highly efficient method for post-processing a large set of patterns. Due to the fact that each pattern is considered exactly once, the complexity of the approach is linear in the number of patterns. On the other hand, this method is highly susceptible to the order that is used to process the patterns, as we already explored in Section 3.2.

As the results of the preceding two sections showed, can orders arising during local pattern mining be used to support the pattern set mining step in data mining. Especially if they are informed by the KDD task that is being addressed, some of the negative influences that arbitrary orders can have are reduced. This opens the door for designing local pattern mining algorithms that impose task-

Table 4.6: **Number of candidate pattern evaluated by** CBA **and** CBC The last column lists the number of patterns for CBC, equating 100%, column two shows the corresponding percentage value for CBA

Dataset	CBA	CBC
Balance (2 Class)	156.01%	99.80 (100%)
Breast-Cancer	127.44%	8179.20 (100%)
Breast-W	52.58%	12913.80 (100%)
Colic	136.29%	73682.90 (100%)
Credit-A	98.49%	65226.50 (100%)
Credit-G	39.14%	155020.90 (100%)
Diabetes	126.52%	3875.80 (100%)
Heart-H	172.57%	14680.00 (100%)
Kr-vs-Kp	15.92%	687715.50 (100%)
Mushroom	95.86%	53615.80 (100%)
Spambase	15.13%	445856.30 (100%)
Tic-Tac-Toe	38.14%	24511.10 (100%)
Voting Record	65.41%	88996.10 (100%)
Average	87.65%	

specific orders during mining.

4.3 Summary

Chapter 4 was concerned with the mining and post-processing of ordered sets of patterns. Similarly to our experiments regarding the effect of different orders on the mining of unordered sets via order-restricted hill-climbing, we explored two different orders in a limited setting in Section 4.1. The difference between the two orders it that one is imposed after mining the set, while the other ones arises from local pattern mining itself, in the form of a top-k. The evaluation shows that the mining-inherent order allows a more efficient use of the resulting pattern sets, enabling the use of a *decision list* classification strategy instead of a more elaborate weighted voting mechanism. It additionally supports an intuitive post-processing step, in which higher-ordered (lower-scoring) patterns are removed.

Pursuing this topic further, we evaluated both of these orders in order-restricted hill-climbing in Section 4.2. Order-restricted hill-climbing is a more sophisticated post-processing method in the way it relates pattern to each other but also vulnerable to the effects of the order used. The retroactively imposed order has been developed for the CBA system (Liu et al. 1998) and is so far the standard in order-restricted pattern set mining in data mining. The order arising from mining shows itself to be better suited for the mining of effective sets, leading to better accuracies of the resulting classifiers. Pattern sets mined in this way are usually also smaller, verifying the phenomena we have observed so far.

Conclusion of Part II

In this part, we discussed a variety of techniques for the post-processing of sets of local patterns into pattern sets that are small, show little redundancy and are effective regarding the tasks in which they are employed.

We split the part into two chapters, considering the mining of unordered and ordered pattern sets, respectively. Unordered pattern sets are very flexible, not relying on attributes such as class labels for the mining process. This makes them well-suited to both predictive and descriptive data mining. The downside is, however, that certain methods of mining for pattern sets cannot be used to process, or produce, unordered pattern sets.

In the first chapter, we introduced two new approaches to post-processing of local pattern mining solutions. Existing methods to post-processing of a local pattern mining were developed in the field of data mining and typically proceed heuristically due to the large number of local patterns that have to be processed. Since the number of patterns is fixed, however, the issue is structurally not different from the one addressed in frequent itemset mining. We therefore suggested an exhaustive algorithm for constraint-based pattern set mining. Given the right constraints, and well-selected thresholds, such a technique is feasible. It allows one to control the properties of derived pattern sets and it guarantees optimality but pays for this with higher computational complexity. The mining operation itself shows all the characteristics of local pattern mining, including the fact that it is not clear how to best choose constraint parameters and a large search space to traverse.

The second technique, which we called Bouncer, performs mining heuristically and is influenced by techniques for feature selection in machine learning. Heuristic techniques can perform efficiently even for large search spaces, and still be effective, mining good patterns to be used, for instance, as features in concept learning. The number of patterns that our proposed algorithms can process lie up to two orders of magnitude above those that exhaustive mining can handle. On the other hand, it cannot be guaranteed that mined pattern sets are optimal in any sense. The only guarantee that can be given when using our techniques is that the resulting pattern set will induce the same partition as the full set. But even this guarantee has to be given up when threshold values are set to find a more compact representation (which often also performs better for classification). In addition to reporting on a technique using order-restricted hill-climbing, we developed Picker*, using upper-bound or-

dered hill-climbing to control the computational complexity. As the evaluation showed, order-restricted hill-climbing is affected by the underlying order, leading to sometimes very different pattern sets that carry similar information. It gives the user more options to control the properties of the solution, however, lying between the full control of the exhaustive search and the upper-bounded search, where only constraint thresholds can be adjusted.

Moving forward with the issue of orders on pattern sets, we considered ordered pattern sets in the context of concept learning in Chapter 4. The advantage is that the order can arise naturally from the mining operation, such as in the case of top-k mining. In Section 4.1, this order was shown to lead to much smaller pattern sets that could be easily post-processed into the basis for an effective classifier. The alternative, an order that is imposed retroactively, can be exploited less effectively.

Both of these orders can, of course, be used as the basis of order-restricted hill-climbing. The retroactively imposed order, in particular, has already been used in some systems for this purpose. Therefore, we evaluated the two orders against each other in Section 4.2 in this context. The results show that as in the case of unordered set mining, the choice of order is very important to the mining result. In both cases, however, effective classifiers were formed, although of rather different size. This is an indication that the heuristic technique is capable of steering the pattern set mining operation to similar optima, even if the underlying pattern sets show different characteristics. The size of the final pattern sets also exceeded that of the result of the exhaustive mining operation on the example data set, illustrating the involved trade-off.

The techniques we have evaluated so far mainly originate from the field of data mining, even if they had not been applied for pattern set mining before, or at least not with those parameters. This is partially a result of the fact that data mining started out as the mining of large sets of patterns and post-processing them is still the most obvious choice for mining sets of patterns. Machine learning, on the other hand, has developed several methods for iteratively mining patterns. The data on which local patterns are mined already reflects the effects of patterns that have been mined in earlier iterations, distinguishing these approaches from post-processing techniques. In the third part of our work, we will therefore evaluate the use of iterative mining techniques for pattern set mining. We will discuss two different paradigms (sequential and parallel mining) and how they reduce redundancy, and evaluate the effectiveness of found pattern sets.

Part III

Iterative Pattern Set Mining

Overview of Part III

In the preceding part of this thesis, we discussed how different post-processing techniques can be used to mine unordered and ordered sets of patterns. Those approaches are decomposed into two distinct phases: 1) mining local patterns and 2) mining a set of patterns from the result of the local pattern mining operation.

A different paradigm, embodied by iterative pattern set mining approaches, is the topic of this third part of our work. As the name already says, pattern sets are mined iteratively in these approaches. More specifically, this means that one or several local patterns are mined, the context of the pattern set mining process modified, and the local pattern mining step repeated. The techniques that we will discuss in this part achieve the modification of the pattern set mining process by manipulating the underlying data, depending on the patterns that have already been mined. A different approach could be to modify the measure that is used for evaluating the merit of a local pattern in combination with other patterns in the set. We will discuss an example for such a technique in greater detail in the concluding remarks of this part.

The first chapter of this part, Chapter 5, is concerned with sequential mining techniques. The underlying assumption in sequential mining is that data that has been described by a pattern (or patterns) is of less importance to later mining iterations. Sequential pattern set mining thus takes the form: 1) mining local patterns, 2) decreasing the importance of covered data, 3) iterating the local pattern mining process. The way in which the decrease in importance is achieved has a strong impact on the resulting patterns, and how they relate to one another.

The strong assumption in sequential mining is that covered data is *unimportant*. In this case instances are removed from further consideration, leading to the sequential covering paradigm. By not explicitly contributing anything to future patterns (implicitly informing them through their absence, however), it becomes almost impossible to find patterns that describe overlapping phenomena. Sequential covering was developed in the context of concept learning and the existing systems use a heuristic local pattern mining step. By replacing this step by an exhaustive approach, we create a new system that we can use to test the effectiveness of sequential covering in directing the pattern set mining process. As we will show in our experimental evaluation, sequential covering as a pattern set mining approach mines effective pattern sets. It can overcome the

pitfalls of local optima that can be reached by heuristic local pattern mining techniques. Additionally, the mined pattern sets are smaller or more accurate (or both) when compared to pattern sets mined by post-processing.

If one makes the weaker assumption that data becomes becomes less important and therefore should be discounted, one arrives at sequential re-weighting techniques. Each instance is given a uniform weight in the beginning and pattern coverage of an instance translates into a smaller weight for future iterations. We will discuss sequential re-weighting in the context of *subgroup discovery*. The system we describe is an adaption of a predictive rule learner, using the same heuristic local pattern mining technique. Subgroup discovery aims at at finding descriptions of the data that are as accurate as possible, however, making suboptimal patterns already undesirable. Once the effects of sequential re-weighting are added to this, a suboptimal pattern will throw future iterations off as well. We therefore replace the heuristic component by an exhaustive approach to evaluate to which degree heuristic techniques lead to non-optimal subgroup descriptions. Additionally, we evaluate the efficiency of both systems to assess whether the use of exhaustive local pattern mining is feasible.

The second chapter of this part, Chapter 6, discusses parallel mining, which leads to tree sets of patterns. The assumption is less strict than in the case of sequential mining: data that has been described is not supposed to be unimportant but instead supposed to be described at a general level, with more specific descriptions of subsets still possible. Parallel mining has been used extensively in concept learning on attribute-value data, in the form of decision tree induction. As we can show, replacing the selection of a single discriminating attribute-value test by an exhaustive local pattern mining allows to directly classify tree-structured data, without recourse to re-encoding of the data. The tree set is also much more compact than classifiers based on unordered or ordered sets of tree patterns. This is yet another setting in which the decomposition of existing pattern set mining techniques allows us to propose an effective system in a principled way. Furthermore, we apply parallel mining in the context of *conceptual clustering* to mine complex descriptions of clusters in a top-down, divisive manner. This direct induction of conjunctive descriptions has not been performed before due to the limitations of the local pattern mining step in existing systems. Finally, exhaustive top-k mining allows us mine more than one pattern per iteration, leading to a completely new type classifier which is related to decision trees and sets of trees.

Chapter 5

Sequential Mining Techniques – evaluating the effects of local pattern mining solutions

In chapters 3 and 4 we discussed several pattern set mining approaches that post-process the result of a local pattern mining operation in a second phase. As we pointed out before, this means that the patterns which are assembled into a set are interesting with regard to the background distribution on the *entire* data set. Pattern set mining as post-processing then evaluates whether patterns are still *interesting* after the effects of other patterns are considered. If there patterns that only *emerge* after these effects are evaluated, those patterns will not be included in the pattern set, simply because the local pattern mining step could not mine them.

An alternative for the formation of ordered pattern sets, which we outlined in Section 2.3, lies in sequentially mining pattern sets. First, local patterns are mined that are interesting with regard to a background distribution. But instead of assembling a very large set of such patterns and then removing redundancies among them, data that has been described by those patterns is now discounted. This leads to a new background distribution against which the interestingness of local patterns is checked. This has two beneficial effects: on the one hand, non-redundancy between patterns from different iterations is ensured in this way. On the other hand, a natural order is created in the pattern set, with patterns from earlier iterations coming earlier in the order. This is different from orders that are imposed subsequently or that arise from the local pattern mining process.

We therefore aim at exploring the advantages (and drawbacks) of sequential mining in this chapter. From the argumentation above we expect pattern sets

mined sequentially to model phenomena more accurately, leading to better performance (and smaller pattern sets). This will likely come at the cost of higher computational complexity, however, since several local pattern mining steps have to be performed, as compared to just one in the case of post-processing techniques.

The first section of this chapter (Section 5.1) is concerned with evaluating sequential covering for the task of mining a pattern set for classification. In sequential covering, data is removed after it has been covered by local patterns. Implicitly, this also means that the patterns found should describe the data as well as possible since no correction in later iterations is possible. Since sequential covering has been developed in the machine learning community, this technique is mostly coupled to the heuristic local pattern mining step with which it was first proposed. As our earlier discussion showed, however, the technique can be decomposed and sequential mining will accept any technique that mines local patterns. As we will show in the experimental comparison of a heuristic and an exhaustive approach to the local pattern mining step, the sets resulting from heuristic and exhaustive local pattern mining perform rather similarly. The composition of the sets is different, indicating that the pattern set mining technique is adept at steering the quality of the entire pattern set towards an optimum with regard to classification acuracy, even if the local patterns are only locally optimal. Additionally, we show that the resulting sets are better suited to the classification task than those created by post-processing. Also, the increase in computational complexity by having to run several local pattern mining processes is not that severe.

In the second section (Section 5.2), the re-weighting mechanism for discounting instances is used to mine a pattern set describing interesting subgroups. In subgroup discovery one cannot simply assume that subgroups are non-overlapping, making soft discounting in the form of re-weighting a better choice than sequential covering. Suboptimal patterns are once again corrected in the context of the entire pattern set. The problem is, however, that while the entire set has merit and subgroups are ordered by their quality according to a measure, *every single* pattern should be as accurate as possible. As the evaluation shows, however, heuristic methods tend to induce suboptimal groups that will influence the sequential process and leads to more errors. Replacing the heuristic method by an exhaustive step allows to mine accurate results without increasing computational cost.

5.1 Sequential covering

The underlying motivation for sequential mining as a pattern set mining technique is to avoid mining patterns that describe subsets that have been described before. A way to guarantee this is to remove the data that has been covered before, which is the approach chosen in sequential covering. A rough outline of sequential covering is given in Algorithm 14, which is an instantiation of Algorithm 12.

Algorithm 14 Iterative pattern set mining

Given: pattern language $\mathcal{L}_p$, data base $\mathcal{D}$, local pattern constraint c, pattern set constraint $\mathcal{C}$
Return: $Th_1(2^{\mathcal{L}_p}, \mathcal{D}, \mathcal{C})$

$\mathbb{S} = \emptyset$
while $\mathcal{D} \neq \emptyset$ **do**
  Mine $Th(\mathcal{L}_p, \mathcal{D}, c)$
  if $\mathcal{C}(\mathbb{S} \cup Th(\mathcal{L}_p, \mathcal{D}, c)) = true$ **then**
    $\mathbb{S} = \mathbb{S} \cup Th(\mathcal{L}_p, \mathcal{D}, c)$
  else
    return $\mathbb{S}$
  end if
  $\mathcal{D} = \mathcal{D} \setminus cov(Th(\mathcal{L}_p, \mathcal{D}, c))$
end while
return $\mathbb{S}$

As we discussed in Section 2.3, the high computational effort that comes with iterative pattern set mining practically enforces that the pattern set mining step is greedy at best. Greedy solutions do not necessarily reach the global optimum and if the local pattern mining step is also performed in a heuristic manner, it is to be expected that the final solution is a very localized solution.

To test this assumption, we will therefore compare in this section the efficiency and performance of two approaches towards ordered pattern set mining using sequential covering:

- CN2$_{\chi^2}$, with a heuristic (beam search) local pattern mining component, inspired by CN2 (Clark and Niblett 1989), the classical sequential covering concept learner.

- CN2$_{CG}$, a newly developed system, in which the local pattern mining component takes the form of top-1 mining

The comparison will be with regard to efficiency and accuracy in the classification setting. The use of heuristic techniques is mainly motivated by an expected reduction of the computational complexity of the entire approach. As we wrote above, however, there is the danger that the combination of a heuristic local pattern mining step with a heuristic pattern set mining technique leads to suboptimal solutions. The experimental evaluation which we perform is therefore designed to shed light on how severe the trade-off between efficiency and effectiveness is. Additionally, we contrast the performance of both algorithms with that of RIPPER (Cohen 1995), the arguably most effective sequential covering concept learner to date.

RIPPER performs the local pattern mining step as greedy hill-climbing search and the pattern set mining as sequential covering. Aware of the dangers of local

optima, however, sophisticated mechanisms have been added to optimize the pattern set further:

1. Each rule, after mining, is pruned by evaluation on a validation set. This is a way of overcoming local optima of the local pattern mining that are due to over-fitting.

2. After the entire set is mined, each rule in turn is potentially replaced. Their replacements are greedily mined from the empty rule and from the current rule by refining it further. The rules are evaluated in the context of the *entire* rule set. In this way local optima of the pattern set mining step can be overcome.

3. If there are instances of the target (positive) class left uncovered, the rule set is extended by a repetition of steps 2 and 3.

RIPPER is therefore a more sophisticated pattern set mining system: not only does it use sequential mining but it also heuristically improves a found pattern set further.

5.1.1 Experimental evaluation

The three main questions to be answered are:

Q5.1 Does the replacement of the heuristic local pattern mining step with an exhaustive version improve the effectiveness (accuracy) of classifiers?

Q5.2 Is the heuristic technique computationally less complex than the exhaustive one?

Q5.3 Does the more sophisticated pattern set mining scheme of RIPPER give it better effectiveness with regard to classification?

The experimental setting for the sequential covering approach are as follows:

- Beam sizes for CN2$_{\chi^2}$ are 5, 10, 20.
- Minimum significance threshold for CN2$_{\chi^2}$ and CN2$_{CG}$ is 3.84.
- RIPPER is run as WEKA's (Frank and Witten 1999) JRIP implementation with default parameters and pruned classifiers evaluated.
- CN2$_{\chi^2}$- and CN2$_{CG}$-classifiers are unpruned.

The data sets and discretization method chosen are the same as in the experimental evaluation in Section 4.2. The classification accuracy is evaluated by ten-fold cross-validation once again.

Table 5.1: **Average accuracy and standard deviation for** $CN2_{\chi^2}$**,** $CN2_{CG}$**, and** RIPPER**.** The left-most column lists data sets, columns 2-4 accuracy estimates for the different techniques (columns 2 & 3 annotated with statistical t-test comparison to RIPPER), $CN2_{\chi^2}$ results are also annotated with the width of the narrowest beam giving rise to the result.

Dataset	$CN2_{\chi^2}$	$CN2_{CG}$	RIPPER
Balance (2 Class)	86.8 ± 3.9 (5) $\circ$	$86.8 \pm 3.9\circ$	80 ± 3.4
Breast-Cancer	81.5 ± 8.4 (10)	80.4 ± 0.77	71.7 ± 0.72
Breast-W	96.4 ± 2.4 (5)	96.4 ± 2.4	95.7 ± 2.1
Colic	82.9 ± 5.5 (5)	88.9 ± 5.9	83.9 ± 7.7
Credit-A	86.5 ± 2.5 (20)	85.8 ± 2.1	85.4 ± 2.5
Credit-G	79.4 ± 6 (10) $\circ$	$79.4 \pm 6\circ$	69.4 ± 5.4
Diabetes	77.4 ± 5.4 (10)	75.1 ± 6.2	76 ± 3.9
Heart-H	83.3 ± 7.5 (5)	81.6 ± 6.3	79.2 ± 7.4
Kr-vs-Kp	94.3 ± 1.4 (5) $\bullet$	$94.3 \pm 1.4\bullet$	99.3 ± 0.4
Mushroom	98.5 ± 0.3 (5) $\bullet$	$98.5 \pm 0.3\bullet$	100 ± 0.0
Spambase	91.4 ± 1.4 (10)	$89 \pm 1.4\bullet$	92.7 ± 1.1
Tic-Tac-Toe	84.6 ± 2.2 (5) $\bullet$	$83.1 \pm 2.2\bullet$	97.1 ± 1.2
Voting Record	95.3 ± 3.2 (5)	96.2 ± 3	95.6 ± 2.8

$\circ$ denotes statistical wins at the 99% level, base-line being RIPPER, respectively

$\bullet$ denotes statistical losses at the 99% level, base-line being RIPPER, respectively

Table 5.2: **Average number of patterns mined by the** $CN2_{\chi^2}$**, and** $CN2_{CG}$**, number of patterns mined and used by** RIPPER

	$CN2_{\chi^2}$	$CN2_{CG}$	RIPPER	
Dataset	# mined	# mined	# mined	# used
Balance (2 Class)	6 ± 1	5.4 ± 2.12	8.7 ± 2.35	5.2 ± 1.22
Breast-Cancer	27.1 ± 6.3	24.6 ± 6.1	11.2 ± 3.97	3.1 ± 0.74
Breast-W	12.8 ± 1.3	11.4 ± 0.8	19.8 ± 2.04	6.6 ± 0.97
Colic	16.5 ± 2.9	8 ± 0.9	12.9 ± 2.23	3.6 ± 0.7
Credit-A	16.2 ± 2.4	14.6 ± 1.84	25.5 ± 2.42	5.8 ± 1.81
Credit-G	31.1 ± 6.1	30 ± 3.37	15.3 ± 2.79	5.5 ± 2.12
Diabetes	10 ± 2.3	11.6 ± 3.37	20.1 ± 3.14	5.2 ± 0.91
Heart-H	9 ± 2	8.8 ± 1.75	10.8 ± 1.39	3.5 ± 0.85
Kr-vs-Kp	3	2	19.2 ± 1.13	15.4 ± 1.27
Mushroom	4.3 ± 0.5	3	8.8 ± 0.79	8.7 ± 0.68
Spambase	10.6 ± 1	5.8 ± 0.92	53.9 ± 3.28	27.3 ± 3.23
Tic-Tac-Toe	6.3 ± 1.3	8.8 ± 0.4	12.1 ± 2.02	10.6 ± 1.27
Voting Record	4.8 ± 0.6	3.2 ± 0.42	8.8 ± 0.63	2.9 ± 1.2

Results $CN2_{\chi^2}$, $CN2_{CG}$, and RIPPER give rise to solutions of similar quality, as can be seen in Table 5.1. RIPPER is significantly better than $CN2_{\chi^2}$ three times (four times *vs* $CN2_{CG}$), χ^2-based optimization being significantly better twice. With $CN2_{\chi^2}$ outperforming $CN2_{CG}$ once, one can conclude that the quality of the heuristically derived classifiers is at least equal to the ones found using $CN2_{CG}$. It should be noted, however, that selecting the right beam size is non-trivial, as the best classification results stem from different sizes for different data sets. The answer to Question 5.1 is, therefore, that the pattern set mining technique corrects the local optimality of the heuristic local pattern mining step to create an effective pattern set.

Two of the cases in which RIPPER finds the better solution are large data sets with rules that have high accuracy on only small subsets (*Kr-vs-Kp, Tic-Tac-Toe*). The reason for this lies in the fact that the penalty that significance measures apply for low frequency at some point outweighs the deviation from the background distribution. The final result is a rather general pattern (relatively high support with good but not great accuracy). By removing the data after covering (and the use of the decision list), no more fine-grained predictor pattern can be found. RIPPER, on the other hand, using additional optimization steps after the first round of sequential covering has been performed, can model such phenomena more accurately. The more sophisticated pattern set mining method gives RIPPER an advantage, compared to the simpler sequential approach the other two algorithms use, which answers Question 5.2.

Since the quality of found rules for the heuristic ($CN2_{\chi^2}$) and complete ($CN2_{CG}$) χ^2 maximization is very similar, the second question focusses on which of the two techniques is more efficient. Table 5.2 shows that $CN2_{CG}$ often (although not always) mines fewer patterns than $CN2_{\chi^2}$. The heuristic

Table 5.3: **Cardinality of pattern sets after order-restricted hill-climbing**

Data set	CBA	CBC
Balance (2 Class)	17.40 ± 5.10	8.00 ± 0.00
Breast-Cancer	98.50 ± 4.74	10.60 ± 1.35
Breast-W	51.40 ± 3.59	12.90 ± 0.57
Colic	88.30 ± 7.79	3.00 ± 0.00
Credit-A	125.00 ± 12.05	2.60 ± 0.69
Credit-G	57.10 ± 10.71	18.80 ± 3.74
Diabetes	59.10 ± 17.82	6.80 ± 1.03
Heart	43.20 ± 2.25	8.30 ± 1.42
Kr-vs-Kp	32.9 ± 1.45	2.20 ± 0.42
Mushroom	26.00 ± 0.00	8.00 ± 0.00
Spambase	2.50 ± 0.97	18.90 ± 1.19
Tic-Tac-Toe	8.00 ± 0.00	10.00 ± 0.00
Voting Record	38.40 ± 2.95	8.00 ± 0.82

and exhaustive mining solutions to local pattern mining therefore mine different patterns, with the pattern set mining technique guiding the search for complementary patterns. The final results are classifiers composed of different patterns, and applied using the *decision list* strategy, that still perform remarkably similar. This phenomenon is consistent with what we could see in the experimental evaluation of order-restricted hill-climbing (Section 4.2.1).

In the context of comparing iterative pattern set mining against post-processing, we reproduce the tables from this experimental evaluation here (Table 5.3). In the first of those two we see varying behavior with regard to pattern set sizes when comparing the two approaches to pattern set mining. When combined with the accuracy results (not reproduced from 4.2.1), iterative mining gives in most cases rise to pattern sets that are smaller, more accurate, or even both. Note again that all classifiers use the same classification strategy. This supports our conjecture that iterative mining leads to more meaningful patterns in the presence of others than post-processing of patterns that have been mined on one and the same distribution.

RIPPER, finally, mines far more patterns in the local pattern mining step but its more refined pattern set mining assembles relatively compact pattern sets. The exception are the three (four) data sets on which it significantly outperforms the simpler sequential covering techniques.

While the size of final pattern sets are an indication of the computational complexity of either approach, a clearer picture should emerge when the number of candidate patterns is considered. Table 5.4 shows, however, that there is no clear winner in this regard. Depending on the beam size and the data set in question, the heuristic approach can be much more or far less efficient than the exhaustive technique. This also indicates that the way of manipulating the underlying data that sequential covering represents is suited to both approaches

Table 5.4: **Number of candidate patterns evaluated by $CN2_{\chi^2}$ (for beam size giving the best solution) and $CN2_{CG}$.** Column three lists number of pattern evaluated by $CN2_{CG}$, equating 100%, column two the corresponding percentage value for $CN2_{\chi^2}$

Dataset	$CN2_{\chi^2}$	$CN2_{CG}$
Balance (2 Class)	139.83%	261.60 (100%)
Breast-Cancer	108.00%	46153.30 (100%)
Breast-W	101.69%	6817.00 (100%)
Colic	2277.10%	2288.70 (100%)
Credit-A	10.76%	1003999.50 (100%)
Credit-G	1.35%	17061266.50 (100%)
Diabetes	40.60%	13030.90 (100%)
Heart-H	36.27%	17809.40 (100%)
Kr-vs-Kp	5.46%	240431.00 (100%)
Mushroom	90.66%	28758.60 (100%)
Spambase	16.14%	2828763.30 (100%)
Tic-Tac-Toe	102.56%	2975.40 (100%)
Voting Record	39.17%	11726.50 (100%)
Average	228.43%	

for the local pattern mining phase. This means that, answering Question 5.3, the exhaustive local pattern mining technique does not need more computational effort to mine the pattern set.

Again, to give some perspective on the mining behavior of iterative and post-processing pattern set mining, we reproduce a second table from the experimental evaluation of order-restricted hill-climbing (Table 5.5). When comparing the computational effort of CBC and $CN2_{CG}$, which are most closely related, one sees that there are six cases where the iterative technique evaluates more candidates than then post-processing one. This is due to the additional mining runs iterative pattern set mining has to perform. However, in another six cases, the outcome is reversed, testament to the easier local pattern mining on smaller data sets whose distributions is more skewed. The numbers for CBA and $CN2_{\chi^2}$ show similar trends.

5.1.2 Conclusions

As our experiments show, sequential covering typically leads to the formation of smaller, more accurate pattern sets for classification than post-processing methods. At the same time, there is no advantage in computational complexity for the post-processing methods. These two characteristics taken together suggest that, given the right KDD task, sequential mining is more attractive in terms of validity (and therefore usefulness) and understandability (due to the smaller pattern sets) than post-processing.

Considering that replacing heuristic local pattern mining methods by ex-

Table 5.5: **Number of candidate pattern evaluated by the complete mining algorithms** The last column lists number of patterns for CBC, equating 100%, column two shows the corresponding percentage value for CBA

Dataset	CBA	CBC
Balance (2 Class)	156.01%	99.80 (100%)
Breast-Cancer	127.44%	8179.20 (100%)
Breast-W	52.58%	12913.80 (100%)
Colic	136.29%	73682.90 (100%)
Credit-A	98.49%	65226.50 (100%)
Credit-G	39.14%	155020.90 (100%)
Diabetes	126.52%	3875.80 (100%)
Heart-H	172.57%	14680.00 (100%)
Kr-vs-Kp	15.92%	687715.50 (100%)
Mushroom	95.86%	53615.80 (100%)
Spambase	15.13%	445856.30 (100%)
Tic-Tac-Toe	38.14%	24511.10 (100%)
Voting Record	65.41%	88996.10 (100%)
Average	87.65%	

haustive ones does not lead to higher computational cost, the use of exhaustive local pattern mining methods seems to be advisable in sequential covering. Our experiments show as well, however, that sequential mining is adept at steering the entire pattern set towards an optimal solution, correcting the suboptimality of local patterns. Especially if usefulness is the dominating criterion, heuristic local pattern mining can therefore be acceptable.

This correction of local optima is a lot easier when it comes to pattern sets that need to have good performance (predictive mining) while *descriptive* patterns should ideally *be* optimal. We will explore the differences in this regard in the next section.

5.2 Sequential re-weighting

The solution that sequential covering finds for mining ordered pattern sets is radical in that data that has contributed to the mining of local patterns once is disregarded in later iterations. In settings such as classification, especially when the final pattern set is used as a decision list, this can be justified. Indeed, as we have seen in the experimental evaluation of the preceding section, different pattern sets can have the same performance. In general, however, it is not clear that local phenomena describe clearly separated subsets of the data. By discounting instances instead of removing them, patterns can be found that partially overlap with regard to the data. This is a sensible solution in terms of *subgroup discovery* for instance, where the descriptions of subsets of known data are of interest, and not only the correct assignment of class labels to unknown instances.

The goal in *subgroup discovery* is two-fold:

1. To find the *best* descriptions of subgroups according to some measure.
2. Not to miss a new subgroup.

The second concern is addressed by the use of sequential re-weighting instead of sequential covering which allows data to contribute that might belong to more than one subgroup. This can already be enough to make sure that the use of a heuristic solution does not lead to very different pattern sets as in the case of sequential covering, when compared to an exhaustive approach. Considering the results we saw in the section on sequential covering, the main question is therefore whether, as is the case in a classification setting, a heuristic solution can be used for the local pattern mining step. Should that not be the case, a heuristic solution might still be admissible for explorative search if it significantly reduced computational complexity. Since data is not removed but only discounted, mining can be expected to be take more effort than in the case of sequential covering so that the computational savings can be larger.

The two algorithms evaluated against each other both use sequential re-weighting for the pattern set mining operation, with the first performing beam search for mining the local patterns combined into the set. This technique was originally introduced as CN2-SD by Lavrač *et al.* (2004). In the second one, the heuristic step is replaced by exhaustive mining for subgroup descriptions, in the form of top-1 mining, creating the novel CG-SD system. Both use *WRAcc* as the interestingness measure.

5.2.1 Experimental evaluation

The evaluation is designed to answer two questions:

Q5.4 Does the heuristic approach (CN2-SD) miss optimal subgroup descriptions?

Q5.5 Is the exhaustive technique (CG-SD) less efficient?

To answer these questions, we perform experiments on a number of UCI data sets, which were selected such that a large range of data cardinality and dimensionality were covered. Numerical attributes have been discretized for the experiments, and we only chose data with discrete classes. Two unsupervised discretization approaches were chosen. In the naïve version, the mean value of an attribute is computed and taken as threshold, leading to two nominal values. For some data sets this leads to very unbalanced value distributions. For these sets, we also chose the threshold in such a way that two roughly equally distributed nominal values result. These data sets are denoted by a trailing "-equal" in the name.

All data is attribute-value data, with the pattern language $\mathcal{L}_{\mathcal{A}}$. We use the class labels as target patterns. While this may seem to be the same task as in the case of concept learning, there are two differences:

Table 5.6: **Comparison for induction of a single subgroup per class value** The first column lists the data set, the last columns the number of candidate pattern evaluated by CG-SD, corresponding to 100%, columns 2–4 the corresponding percentage-values for different settings of CN2-SD.

Dataset	CN2-SD$_{20}$	CN2-SD$_{10}$	CN2-SD$_5$	CG-SD
Balance-2-Class	644.00%	436.00%	278.00%	50 (100%)
Breast-W	3443.04%	1791.14%	948.10%	79 (100%)
Breast-W-equal	3061.36%	1609.09%	856.82%	88 (100%)
Car	1722.22%	898.08%	481.61%	261 (100%)
Colic	10569.26%	5336.49%	2723.65%	296 (100%)
Colic-equal	10699.64%	5394.31%	2772.95%	281 (100%)
Credit-G	2106.84%	1062.73%	541.76%	1492 (100%)
Credit-G-equal	2036.56%	1028.06%	523.89%	1436 (100%)
Diabetes	2445.24%	1329.76%	705.95%	84 (100%)
Diabetes-equal	1014.78%	550.25%	291.63%	203 (100%)
Heart-H	3682.01%	1876.19%	976.72%	189 (100%)
Heart-Statlog	2639.30%	1342.36%	696.07%	229 (100%)
Heart-Statlog-equal	2416.27%	1227.38%	637.70%	252 (100%)
Krkopt	1463.92%	765.52%	413.20%	2697 (100%)
Mfeat-Morpho	2090.53%	1244.44%	672.43%	243 (100%)
Mfeat-Morpho-equal	2086.42%	1249.38%	676.95%	243 (100%)
Nursery	3283.92%	1692.60%	888.75%	311 (100%)
Segment	7784.20%	3949.24%	2015.46%	595 (100%)
Tic-Tac-Toe	1717.58%	879.69%	461.33%	256 (100%)
Voting Record	7201.55%	3655.04%	1883.72%	129 (100%)
Zoo	13206.91%	6714.63%	3400.96%	1982 (100%)
Pendigits	5523.76%	2800.83%	1313.24%	846 (100%)
Mushroom	11928.74%	5997.13%	3074.71%	522 (100%)
Average	4155.07%	2119.10%	1090.17%	

- The goal is not the mining of well-classifying rules but instead the mining of good descriptions for the classes.

- Since *WRAcc* is asymmetrical, each class has to be considered in turn to find good descriptions.

For CN2-SD, beam sizes 5, 10, and 20 were evaluated.[1]

The first table, Table 5.6, shows the number of candidate patterns evaluated by the two techniques for the induction of the single highest-scoring subgroup description. Although the heuristic approach to local pattern mining always finds the globally optimal pattern, the computational complexity is also consistently higher than that of the exhaustive top-1 search.

[1] 20 was suggested by a reviewer, 5 and 10 evaluated as well as to not bias the efficiency estimation against CN2-SD

Table 5.7: **Comparison of a complete *subgroup discovery* run** First column lists the data set, last columns the number of candidate pattern evaluated by CG-SD, corresponding to 100%, columns 2–4 the corresponding percentage-values for different settings of CN2-SD.

Dataset	CN2-SD$_{20}$	CN2-SD$_{10}$	CN2-SD$_5$	CG-SD
Balance-2-Class	537.37%	356.48%	214.86%	471 (100%)
Breast-W	1588.41% •	865.23% •	442.74% •	179625 (100%)
Breast-W-equal	1475.91% •	800.42% •	504.28% •	117251 (100%)
Car	689.71%	350.11%	184.15%	61609 (100%)
Colic	1109.36% •	563.09% •	285.68% •	395291 (100%)
Colic-equal	892.98% •	458.43% •	218.20% •	476363 (100%)
Credit-G	231.50% •	113.24% •	55.72% •	543376 (100%)
Credit-G-equal	152.01% •	66.88% •	32.22% •	684859 (100%)
Diabetes	1948.31% •	1061.99% •	486.26% •	8030 (100%)
Diabetes-equal	316.79% •	169.46% •	88.00% •	13836 (100%)
Heart-H	1223.74% •	617.57% •	391.68% •	22415 (100%)
Heart-Statlog	1263.71%	911.69% •	479.43% •	5509 (100%)
Heart-Statlog-equal	1178.30%	595.25%	304.50%	6692 (100%)
Krkopt	655.85% •	337.64% •	182.00% •	394671 (100%)
Mfeat-Morpho	1724.76%	1017.61%	530.07%	11775 (100%)
Mfeat-Morpho-equal	1465.44%	864.24%	450.13%	12052 (100%)
Nursery	2082.05%	1066.15%	552.82%	975 (100%)
Segment	5610.12%	2835.89%	1410.76%	19293 (100%)
Tic-Tac-Toe	771.75%	391.34%	201.31%	2818 (100%)
Voting Record	1500.65% •	2454.68% •	2540.43% •	3643096 (100%)
Zoo	13206.91%	6714.63%	3400.96%	1982 (100%)
Pendigits	2705.73% •	1443.45% •	733.19% •	279217 (100%)
Mushroom	10002.78%	4870.87% •	2377.40% •	8900 (100%)
Average	2210.15%	1150.94%	588.10%	

• denotes that a non-optimal subgroup description, that is a description having a lower score than the highest possible, has been found

The second table, Table 5.7, reports the number of candidate patterns that were evaluated during a *complete* run, that is a run that ends only when all instances have been covered by at least one pattern. Once additional local pattern mining phases are part of the set mining operation, smaller beam sizes lead to better computational efficiency. However, at the same time, for all settings in which CN2-SD has lower computational complexity, in at least one iteration, a non-optimal subgroup description is mined. That means that the heuristic returns a pattern with a lower score than maximally possible in that iteration. This tendency holds even for a beam size of 20 on many data sets. Given the process of iterative set mining, mining a non-optimal subgroup description affects the subsequent data manipulation and therefore the appropriateness of patterns that are mined later. Additionally, even though on several data sets smaller beam sizes lead to fewer candidate patterns that are evaluated by CN2-SD, on average the exhaustive search has lower complexity.

The answer to Question 5.4 is therefore that the heuristic technique *does* miss optimal subgroups, distorting the final pattern set. Additionally, exhaustive mining is not more expensive, answering Question 5.5. These findings also mean that when it comes to pure performance of a mined pattern set, as in the classification setting discussed before, using heuristic local pattern mining in combination with the greedy approach to ordered pattern set mining is feasible. When the goal is descriptive, however, heuristic techniques run the risk of mining incorrect descriptions and propagating this effect by the data manipulation employed.

5.2.2 Conclusions

Considering that sequential re-weighting was introduced in the context of *subgroup discovery*, we evaluated the impact of using heuristic techniques in the local pattern mining step. In classification, good performance is the main characteristic of a "good" pattern set, which allows the iteration technique to smooth out non-optimal individual patterns by inducing complementary patterns in later iterations. In descriptive mining, however, the aim is to find accurate and interpretable descriptions; usefulness has therefore more to do with validity and understandability. This makes non-optimal patterns less attractive, especially given the distorting effect they have on later iterations, suggesting that for sequential re-weighting for descriptive tasks the use of exhaustive local pattern mining is advisable.

5.3 Summary

In this chapter, we considered sequential mining techniques in the context of classification and subgroup discovery. Sequential covering can correct locally suboptimal patterns to allow us to mine a pattern set approaching optimality. Additionally, these pattern sets are of smaller cardinality than ones mined by post-processing methods, making the full pattern set more understandable. This

kind of correction is not always possible in descriptive tasks, however. The result that exhaustive local pattern mining techniques do not lead to a more expensive pattern set mining process than the use of heuristic ones is therefore promising.

Both techniques discussed in this section, sequential covering and sequential re-weighting, make an explicit assumption that patterns describe a complete subset of data rather well. While this assumption is weakened in sequential re-weighting, its main focus is still on patterns that *do not* describe a subset of already covered data in more detail. It can be argued however, that this is still too strong an assumption since data might have different characteristics depending on the granularity at which it is considered. An alternative to sequential mining, and the ordered pattern sets it produces, is therefore parallel mining which leads to the formation of tree-sets. Tree-sets allow coarse descriptions of data that are iteratively refined so each subset is not described by *one* but by *several* patterns in the set. We will discuss tree sets, parallel mining, and the tasks for which they can be mined in the next chapter.

Chapter 6

Exploring the Power of Parallel Mining Techniques

The main difference between tree and ordered sets lies in the greater flexibility of the former. Since the order defined on tree sets is only a partial order, they can be restricted further to totally ordered sets but, if the data suggests, also model less stringent dependencies between patterns in the set. In contrast to ordered set mining, not only data uncovered by a pattern should be considered in further iterations but also the data that has been covered already.

In terms of iterative mining, a good technique for doing this lies in splitting the data: since the matching function is binary, two subsets are formed. New patterns are then mined on each subset, which are split in turn after the mining. This leads to a total order of patterns regarding predecessor and successor patterns while the order regarding patterns that are mined on the other subset is only partial. Originating in decision trees, in which only single attribute-value tests are used to split the data, this technique has been shown to induce effective and often relatively compact classifiers.

There are two main effects that stem from these characteristics: tree sets evolve into ordered sets if an ordered set (as created by sequential covering) is the accurate way to describe the data. Furthermore, since patterns do not have to model data in a single step, they need to be less specific, possibly leading to smaller pattern sets. In this chapter, we therefore compare tree sets to other types of pattern sets to see whether these advantages materialize.

Parallel mining was originally introduced in the context of decision tree induction and is, similarly to sequential mining, usually discussed together with a particular local pattern mining method. As shown in Section 2.3.3, however, it can be described in general terms and the local component can be instantiated in a variety of ways. We use this chapter, therefore, to explore ways to leverage the parallel mining paradigm for the mining of compact and effective sets for different tasks.

In the first section of this chapter, Section 6.1, we illustrate tree pattern

set mining in the context of *conceptual clustering*. This is a rather natural formulation since clusters are sets of instances that are *similar* to each other. As a quick check in the real world will show, however, similarity is largely a product of contrast with other groups. Men are more similar to each other on a superficial level than to women but once one considers only men, that similarity is far less obvious. Successively more fine-grained descriptions capture clusters therefore better than one-shot descriptions. Clustering trees have been induced before, the patterns were of low complexity, however, to keep local pattern mining efficient. By replacing the local component by an efficient exhaustive miner, our system can induce more complex patterns. The experimental evaluation shows that the resulting pattern set gives a compact description of the formed clusters in the pattern language while maximizing intra-cluster similarity in a heuristic manner overall. When compared to clusters that are formed first and then described, the description given by the tree sets is far less complex *and* more accurate.

In the second section, Section 6.2, we use tree set mining on tree-structured data for concept learning. Tree structured data is harder to mine than conjunctive patterns or unstructured patterns like itemsets. The usual approaches to classification either rely on large sets of predictive patterns like the ones we evaluated in Section 4.1, or on the re-encoding of data in terms of patterns and, for instance, decision tree induction. Especially if there is a need to understand which properties of trees the classifier uses to make a prediction, however, a small sets of patterns with a clear relationship is desirable. By mining for tree patterns directly in each iteration, such a set can be assembled. As the evaluation shows, are the resulting pattern sets are smaller than ordered pattern sets or tree sets which are formed in a post-processing step. At the same time they give comparable results in terms of predictive accuracy.

Finally, in Section 6.3, we increase the number of patterns mined in each iteration. What we are aiming to achieve in this way is to make the one-shot mining per iteration more flexible. Similarly to how sequential re-weighting allowed data to contribute to more than one pattern compared to sequential covering, several patterns per iteration allow data to contribute to more than one pattern per generality level. This is a new approach in that existing approaches to decision tree formation, even the ones we discuss in the other two sections, rely on a single pattern for splitting the data. Extending tree sets in such a way comes at a price, however. On the one hand, this necessitates the formulation of a more complex matching function. On the other hand has the predecessor-successor-relation to be augmented by an order relation among the patterns of *each* iteration. This last section is therefore more extensive than the other two. As we show in its experimental part, however, extending tree sets in such a way pays off in terms of complexity and accuracy.

6.1 CG-Clus – tree sets for conceptual clustering

We have already characterized clustering as the task of partitioning the data (cf. Section 1.2.3). The guiding principle for this partition is that ideally instances in a cell are highly similar to each other (intra-cluster similarity). Additionally, instances assigned to different cells should show little similarity (inter-cluster dissimilarity). For the numerical case clusters can be represented for instance by centroids or medoids and the similarity quantified by a vector norm such as the L1 or L2 norm.

Performing the *clustering* task for instances described by nominal attributes is more difficult since a similarity measure is often harder to define. In general, instances are considered similar if they agree on the values of many attributes. One measure for judging the quality of a set of clusters is *Category Utility* (Gluck and Corter 1985) but others have been defined in the literature as well. The goal of *conceptual clustering*, is to produce a description of the found clusters in terms of the conceptual language used to describe the instances.

According to Hoepner (Höppner 2004) clusters can be considered as deviations in distribution from a default (or background) distribution w.r.t. certain attributes. Therefore, top-1 mining can be used to find a pattern that separates subsets with regard to those distributions, the data split and more refined patterns mined in subsequent iterations.

Since the goal, as mentioned above, is similarity in as many attributes as possible, *all* attributes are considered targets, with the symmetric measure CU leading to the induction of patterns in whose coverage space either the occurrence of *true* or false values will be higher than expected. The multi-dimensional extension to the top-k mining technique we described in Section 2.1.1 will be used for local pattern mining phase within the general parallel mining algorithm. We term this algorithm CG-Clus. This is to the best of our knowledge the first time that complex correlation based patterns are used in divisive clustering approach. Another approach that uses parallel mining to form clustering trees, that we are aware of, exists in (Blockeel et al. 1998). The patterns used in this approach are of relatively low complexity, however, staying close to the original decision tree paradigm.

To evaluate CG-Clus, we shall compare it to Cobweb (Fisher 1987), the arguably best-known conceptual clustering technique, which also produces dendrograms. Cobweb iteratively and directly processes instances, using four operators: assigning an instance to an existing dendrogram node, creating a new node, splitting an existing node, or merging two existing nodes. Cobweb's dendrograms are tree sets of conditional probability vectors which refer to attributes' values. The further removed from the top-node one is, the more skewed those distributions become, culminating in nodes describing single instances (or a set of identical instances). The dendrogram constructed in this way essentially includes further and further refined patterns.

A different approach to finding clusters described by conjunctive descriptions

is *conceptual cluster-mining* (Perkowitz and Etzioni 1999), where a clustering algorithm is used to find a clustering. Each cluster is then treated as a single class, and conjunctive concepts learned on them in a post-processing step. Afterwards, all instances matching a concept are considered to be in one cluster, possibly producing overlapping clusters. While the language describing the clusters is $\mathcal{L}_\wedge$ (Definition 1.1.3), the patterns expressed in this language are not mined directly under consideration of their partitioning effect. Since COBWEB does not produce conjunctive descriptions, we additionally compare against a cluster mining technique using AUTOCLASS (Cheeseman et al. 1988) and RIPPER.

6.1.1 Experiment 1: COBWEB vs CG-CLUS

COBWEB's direct assignment, and the fact that it uses a rather flexible pattern language, gives it great flexibility in assembling clusters. On the other hand does its iterative nature (processing instances one after another without backtracking) expose it to potential ordering effects. This leads directly to the first question,

Q6.1 Do COBWEB's clusterings differ strongly from CG-CLUS' in terms of *CU*, and in actual composition of the clusters?

If COBWEB produces higher *CU*-scores, which means better intra-cluster similarity and inter-cluster dissimilarity, the next question is whether this effect is connected to the more flexible description:

Q6.2 Are conjunctive descriptions of COBWEB clusters more complex than the ones mined directly by CG-CLUS?

To clarify this question – it is of course possible to find a conjunctive description for just about any group of attribute-valued data, especially if that data is relatively homogeneous with regard to the attribute values, which would be measured by the *CU* value. However, the minimal amount of conjuncts needed to distinguish such a group from others can vary, again depending on the heterogeneity, and the accuracy with which the description matches the instances of the group as well. Long descriptions of weak accuracy are not useful to a user.

Experimental setup To compare the agreement of two clusterings, we use the *Rand* index, as defined in Section 3.2.2. If the number of clusters in a clustering is different, the *Rand* index will obviously show a dissimilarity. Therefore we attempted to form a number of clusters corresponding to the number of class values in each data set in our experiments.

To obtain a given number of clusters from a COBWEB dendrogram, there are two possibilities – the user selects certain nodes in the tree, disregarding the structure underneath them, or the growth of the dendrogram is limited.

In the first case, the fact that COBWEB often constructs dendrograms in which every instance is sorted into its own cluster, makes this a non-trivial

procedure. Also, selecting all nodes from the same level of the tree does not guarantee a good solution *CU*-wise. In the case of an agglomerative or divisive clustering algorithm, one can stop the merging (or splitting) of clusters once their distance exceeds (falls below) a certain threshold or once a given number of clusters is reached.

This is not as simple for COBWEB's iterative instance processing. Instead, in the WEKA implementation, a minimum *CU*-gain can be set which determines whether new nodes in the dendrogram are introduced, or existing nodes split. By starting out with a lenient threshold, systematically tightening it when more than the desired number of clusters is formed and relaxing it when the tightening proved to be too strict, it is possible to approximate the desired number of clusters.

Unfortunately, this method does not always guarantee obtaining the target number of clusters, since COBWEB sometimes forms just one cluster or tens, or even hundreds, depending on a 0.0001 difference in the threshold value. Instead of arbitrarily merging clusters, we used the COBWEB-solution whose number of clusters is closest to the actual number of classes, unless this number is 1, i.e. all instances were sorted into the same cluster. After determining the COBWEB-clustering, we attempted to construct the same number of clusters using CG-CLUS.

As for our technique, CG-CLUS, to obtain the desired number k of clusters, the $k-1$ best splits are used. In this sense, it is very much a divisive clustering algorithm in the classical sense. Since a good *CU* score on a small subset is easier to achieve than on a larger one, patterns' scores are weighted with the proportion of instances of the complete set that they were derived on. The resulting dendrogram is decision tree-like in the way data is split on patterns and their impact discounted on the population size.

This means that the pattern set mining is performed as top-k mining, with the measure taking the form:

$$CU(sp(p, \mathcal{D}_{sub})) \cdot \frac{|\mathcal{D}_{sub}|}{|\mathcal{D}|}$$

$\mathcal{D}_{sub}$ denotes for the subset on which p has been mined. The mining process is stopped once no pattern can be mined on any current subset that is included in the top-k.

Owing to the need for binary attributes, discretization was performed as in the *subgroup discovery* experiments. Nominal attributes were binarized by turning each attribute A with values $v_1, \ldots, v_n$ into n binary (values $\{t, f\}$) attributes $A = v_i, i \in [1, \ldots, n]$. Due to aforementioned ordering effects, we performed 10 restarts of COBWEB on randomly permuted data sets, and averaged *CU* and *Rand index* values.

Results To answer the first question asked above, we report *CU*-values and the Rand-index for CG-CLUS- and COBWEB-clusterings for a variety of data sets in Table 6.1. For the data sets for which hundreds or even thousands of

Table 6.1: *CU* of the CG-CLUS clusterings, and *CU* of COBWEB's solution, averaged over 10 runs, Rand-index of the two clusterings

Dataset	CU_{CG}	CU_{CW}	Rand
Credit-G	0.4408	0.0239 ± 0.002	N/A
Credit-G-Equal	0.4753	0.1161 ± 0.0293	N/A
Kr-vs-Kp	0.5343 ± 0.0040	0.5782 ± 0.0012	0.7817 ± 0.0029
KrkOpt	0.1536 ± 0.0072	0.1369 ± 0.002	0.7396 ± 0.028
Letter	0.1742 ± 0.0075	0.1342 ± 0.0151	0.7629 ± 0.0275
Letter-Equal	0.1677 ± 0.0053	0.1439 ± 0.0063	0.8759 ± 0.0001
Mfeat-Fourier	0.4743	0.4487 ± 0.1334	N/A
Mfeat-Fourier-Equal	0.7183	0.1855 ± 0.0289	N/A
Mfeat-Karhunen	0.457	0.0203	N/A
Nursery	0.3555	0.0846 ± 0.0273	N/A
Optdigits	0.4609 ± 0.0029	0.5234 ± 0.0229	0.7865 ± 0.0148
Optdigits-Equal	0.6865 ± 0.0203	0.7936 ± 0.0565	0.8509 ± 0.0069
Pendigits	0.4336 ± 0.0091	0.4015 ± 0.0074	0.8519 ± 0.0004
Segment	0.5878 ± 0.0122	0.5438 ± 0.0505	0.7994 ± 0.1029
Segment-Equal	0.7925 ± 0.0083	0.7916 ± 0.0063	0.8984 ± 0.0039
Waveform	0.9791	1.1624 ± 0.0229	0.7822 ± 0.0192

clusters were formed by COBWEB we did **not** attempt to form the same number of clusters using CG-CLUS. Instead we report on the *Category Utility* of the "correct" solution of CG-CLUS (i.e. the clustering having as many clusters as classes in the data), the average *CU* for COBWEB and no value for the *Rand*-index.

The resulting *Category Utilities* show that far from always giving rise to superior scores by using the more flexible clustering scheme, the quality of COBWEB's solution is clearly affected by ordering effects in the data. If the right ordering of instances is used, COBWEB constructs very good solutions, if not, *CU* values are rather low or the dendrogram is huge. When threshold differences of 0.0001 make the difference between a single cluster and a dendrogram having hundreds of leaves, it is difficult for the user to make an informed decision on which clusters to merge. For data sets where a reasonable number of clusters was constructed, COBWEB's average *CU* is greater than that of CG-CLUS four times, less six times, while at the same time exhibiting similarities of the clusterings in excess of 0.7. This means that Question 6.1 has to be answered negatively, COBWEB does not always translate the greater flexibility of its assignment mechanism into better *CU* values. Also, the solutions are rather similar.

Table 6.2 is used to give insight into the second question. It lists the number of classes in the data, and the average number of leaves in COBWEB's dendrogram over ten runs. Each of these leaves is then considered a class and rules learned for each class, using RIPPER (unpruned). The average number of rules and their recovery rate, that is to say their accuracy on the training date they

Table 6.2: Classes per data set, average number of clusters formed by COBWEB, number of conjunctive rules learned on the clustering using RIPPER, recovery rate (that is training set accuracy of learned rules)

Data sets	# of Classes	# of Clusters	# of Rules	Recovery rate
Credit-G	2	349.5 ± 180.96	38.9 ± 8.86	$65.44\% \pm 17.61$
Credit-G-Equal	2	202.5 ± 191.02	42.7 ± 8.3	$78.34\% \pm 18.8$
Kr-vs-Kp	2	2.5 ± 0.7	15.7 ± 14.47	$99.74\% \pm 0.3$
KrkOpt	18	13.8 ± 5.2	14.5 ± 5.98	$99.99\% \pm 0.003$
Letter	26	18.7 ± 13.01	139.6 ± 40.65	$98.71\% \pm 0.64$
Letter-Equal	26	24.5 ± 5.23	211.2 ± 24.04	$97.41\% \pm 0.61$
Mfeat-Fourier	10	54.9 ± 60.58	51.9 ± 24.82	$95.13\% \pm 3.84$
Mfeat-Fourier-Equal	10	58 ± 24.35	24.6 ± 2.87	$96.99\% \pm 1.2$
Mfeat-Karhunen	10	663.7 ± 173.25	86.3 ± 19.59	$64.28\% \pm 8.75$
Nursery	5	1480.8 ± 958.07	1102.3 ± 710.23	$97.22\% \pm 0.92$
Opdigits	10	8.5 ± 1.65	103.4 ± 16.59	$93.33\% \pm 1.81$
Optdigits-Equal	10	8.6 ± 1.17	111.2 ± 17.21	$93.47\% \pm 1$
Pendigits	10	7.2 ± 0.63	60.5 ± 7.01	$99.73\% \pm 0.08$
Segment	7	4.5 ± 0.7	6.6 ± 1.17	$99.99\% \pm 0.01$
Segment-Equal	7	6.2 ± 0.42	38.9 ± 3.24	$99.42\% \pm 0.13$
Waveform	3	3	52.4 ± 2.75	$97.31\% \pm 1.04$

were derived on, is also reported. It should once again be noted that CG-CLUS builds a tree set of conjunctive descriptions for clusters – which have 100% recovery rate. Furthermore, it can easily be constrained to form as many leaves as classes, and therefore desired clusters, exist in the data.

The experiments show that COBWEB rarely forms a number of clusters that corresponds to the number of underlying classes. In addition, most of the time far more rules than classes are learned on the data. The recovery rate is close to one hundred percent in all but three cases. Note also that in COBWEB's original cluster representation each dendrogram node (whose number exceeds that of the leaves) will have its own vector of conditional probabilities. This means that in most cases the description of clusters (the main goal of conceptual clustering) will be rather complex; more complex and harder to constrain than if clustering is performed as iterative pattern set mining. We can therefore answer Question 6.2 positively, since COBWEB's descriptions are more complex.

6.1.2 Experiment 2: CONCEPTUAL CLUSTER-MINING vs CG-CLUS

As we outlined in the introduction to this section, *cluster-mining*, the approach proposed in (Perkowitz and Etzioni 1999), forms clusters and their conceptual descriptions in separate steps. In the first step, a clustering algorithm is used to partition the data. In a second step, each cluster is considered a separate class, and conjunctive descriptions of the clusters learned. Finally, instances are redistributed in such a way that they match the descriptions. While instantiating their framework with particular selections for the clustering algorithm (PAGEGATHER), and concept learner (RIPPER gives the best results), it allows

different choices as well. PageGather has been explicitly developed to cluster web pages. We therefore replace it with Autoclass, an effective clustering algorithm that is capable of dealing with nominal data.

Autoclass is based on Bayesian principles and thus not directly optimizing *CU*. On the other hand, it has once again greater flexibility than CG-Clus in assigning instances *directly* to clusters – not indirectly via the found description. In addition, decoupling the processes of forming clusters and finding a description gives the actual concept formation greater flexibility than CG-Clus possesses. Since a *cluster-mining* approach has greater flexibility (and make conjunctive concept formation a non-integral part of the mining process), similarly to the first setting, the questions are:

Q6.3 How similar are CG-Clus' and Autoclass' clusterings?

Q6.4 How complex are conjunctive descriptions of Autoclass' clusters, compared to CG-Clus' ones, and how much information about the underlying instances is recovered?

Experimental Setup Autoclass can be supplied with the number of clusters it should create. For each data set, Autoclass performed 250 restarts with 200 iterations each. The assumed model was single multinomial for all attributes. We used the best clustering found for comparison with the pattern set mining approach.

Regarding the cluster mining solution using Autoclass and Ripper, Table 6.3 lists the number of classes per data set (both CG-Clus and Autoclass form the same number of clusters), the number of rules learned by Ripper and their recovery rate on Autoclass' solution, as well as the *Rand*-index of the two clusterings.

The similarity of the clusterings produced by the two methods is generally very high, with three exceptions, answering Question 6.3. Of particular interest is that the Rand-index is generally considerably higher than when we compared CG-Clus and Cobweb. Autoclass' solutions give rise to smaller descriptions than Cobweb's do, and Autoclass' rules achieve a very high recovery rate on *all* clusters. On the other hand they still exceed the number of clusters (and thus patterns in CG-Clus' tree) by far. This means that while being rather close in actual composition, the description of Autoclass' solution is far more complex. The answer to Question 6.4. is therefore that the Cluster-mining technique produces more complex description that are very accurate.

To summarize, while being less flexible in forming clusters, the novel system CG-Clus finds clusterings that are highly similar to the solutions of two more flexible schemes that are well-established in the literature. The mining process itself guarantees high intra-cluster similarity in a single run of the algorithm and in addition, the formulation of conjunctive cluster descriptions of low complexity.

Table 6.3: Classes per data set, number of descriptive rules found by RIPPER on the AUTOCLASS-solution and their accuracy, and similarity of found clusterings

Data sets	# of Classes	# of Rules	Recovery rate	Rand
Credit-G	2	7	100%	0.5012
Credit-G-Equal	2	25	99.1%	0.558
Kr-vs-Kp	2	2	100%	0.9185
KrkOpt	18	31	100%	0.9663
Letter	26	268	98.86%	0.9016
Letter-Equal	26	277	97.16%	0.9402
Mfeat-Fourier	10	92	99.15%	0.8673
Mfeat-Fourier-Equal	10	64	99.4%	0.8481
Mfeat-Karhunen	10	89	99%	0.8729
Nursery	5	5	100%	0.6543
Optdigits	10	117	95.53%	0.8656
Optdigits-Equal	10	115	96.89%	0.8887
Pendigits	10	73	99.66%	0.9019
Segment	7	9	99.91%	0.8345
Segment-Equal	7	10	99.65%	0.9002
Waveform	3	50	98.2%	0.7877

Related work

Our description of related work differs in this chapter from the ones preceding it for the simple reason that iterative pattern set mining has, to our knowledge, not been used for conceptual clustering before.

One of the earliest approaches to inducing conjunctive descriptions of conceptual clusters is the CLUSTER/2 system (Michalski and Stepp 1983). While CLUSTER/2 works bottom-up in a heuristic manner by generalizing pairs of *seed instances*, our technique induces concepts top-down and gives guarantees with regard to the quality of found solutions. The *conceptual cluster mining* task, introduced by Perkowitz *et al.* (1999), is similar to CG-CLUS in terms of the final description. Their goal is to induce clusters that are cohesive but also describable by a simple concept. To this end, they use their PAGEGATHER system for clustering webpages and RIPPER to learn the concept separating each cluster from all others. The final solution consists of all subsets of instances that correspond to the learned concepts. Since both the clustering and the rule learning algorithm could be instantiated differently, the *conceptual cluster mining* framework is rather general. The approach does have potential drawbacks as discussed in the experimental evaluation.

In (Nevins 1995), an incremental branch-and-bound clusterer for the formation of hierarchies was introduced. Since addition of new observations can have a severe effect on the existing hierarchy, re-insertion of instances and clusters is performed during the formation process. The algorithm is language independent and the information about the relationship between instances expressed only via a similarity measure. To restrict the number of evaluation steps needed, the set

of nodes that could act as parents in the hierarchy to the instance or cluster to be inserted is limited. To this end, an upper bound on the amounts of information which *can* and *cannot* be accounted for by potential parent nodes is calculated. This best value is based on the evaluation of picking this node's parents as parents of the new instance or cluster.

The measure we used for the quality of cluster descriptions in this work is *Category Utility*, with the probably best-known clusterer using this measure being COBWEB. Due to its incremental instance processing, the ordering of instances has an effect on the solution. To address this effect, Fisher (Fisher 1996) explores several re-distribution and re-clustering techniques for greedily improving an existing clustering. We find the optimally discriminating patterns in the first run instead. A second issue addressed in Fisher's work is related to the effect we observed in the experimental evaluation, namely that COBWEB on certain data sets tends to create a large amount of clusters, gaining only a small increase in *Category Utility*. The solution discussed in (Fisher 1996) is similar to post-pruning in decision tree learning in that certain branches of the clustering hierarchy are removed during validation on a separate data set. Third, Fisher discusses possible shortcomings of *Category Utility* as a quality measure for clusterings. Assuming that a clustering is used for classification afterwards, he suggests properties such as number of leaves, maximum path length, branching factor and classification cost, e.g. number of attributes to be evaluated, for measuring the quality of a clustering tree. The formulation of *conceptual clustering* as an iterative pattern mining task allows for easy inclusion of such criteria for pre-pruning. Finally, possible alternatives to *Category Utility* mentioned in this work could be used in top-1 mining if they are convex.

Potentially the main difference between existing *conceptual clustering* techniques and our approach lies in that the two main aspects are reversed. Those two main aspects are data manipulation (assignment to different clusters) and derivation of the cluster descriptions. Other approaches first form clusters and derive the descriptions implicitly (or as a post-processing phase). In contrast, CG-CLUS first enumerates descriptions (patterns) while evaluating their effects on cluster coherence, and *then* forms subclusters for the next mining iteration. A different view of this would be that improving cluster coherence and finding descriptions are coupled, different from other approaches.

A conscious approach towards using decision tree techniques for clustering is the TIC system (*T*op-down *I*nduction of *C*lustering trees) (Blockeel et al. 1998). In this work, each node in the tree consists of the test that minimizes a distance measure for instances sorted into each of the two subsets. The final result is a dendrogram whose leaves can be described by the conjunction of all decision nodes on the path to the leaf. Although the authors do not refer to their work as *conceptual clustering*, they locate it firmly in the field by drawing connections to for instance the COBWEB system. While the language they use is more expressive (first-order logic), there is no induction of complex tests in each note, as in our approach.

The authors on whose work our formulation of the top-k mining problem is based, Morishita *et al.*, developed their ideas into an abridged version of our so-

lution in (Sese and Morishita 2004). Using the property that interclass-variance is a convex function as well, they derive clusters that show high similarity in gene expression levels and are described by conjunctions of nominal attributes. They omit the iteration step, however, content with listing a top-k list of partitioning-options based on a single split. This could however be extended to an algorithm resembling ours, as could their top-k clusters solution form the basis for something resembling beam search for the best partitions.

6.1.3 Conclusions

The new system, CG-Clus, allows the direct mining of conjunctive descriptions for conceptual clusters, inducing a dendrogram along the way. This is in our opinion the preferable option to a post-processing approach like cluster-mining in which patterns are formed after the cluster formation. Depending on the task, conjunctive patterns may not be the best choice, but even in those cases, a pattern set mining approach gives a certain control over the number and quality of clusters. Thus, CG-Clus is a system than can be used as an alternative to or augment existing techniques.

6.2 Tree^2 – tree sets of tree patterns for classification

After discussing iterative pattern set mining in a context in which it had not been used before, and showing its effectiveness, we return to concept learning. We use tree-structured data to compare pattern set mining using exhaustive search in each iteration to build tree sets of tree-structured patterns, leading to the name Tree^2 for our technique. Usually, classifiers for tree-structured data take the form of predictive tree patterns, as in XRules and CtC (cf. Section 4.1), or patterns are mined, the data re-encoded and machine learning techniques trained on the new representation. Tree-structured patterns are more expensive to mine, both in terms of enumeration and in terms of matching them against instances, than unstructured languages like conjunctive patterns or itemsets. One goal is therefore the induction of *small* pattern sets, which should be possible with parallel mining. This has a positive side-effect in that tree-structured data is often not easily understood by human observers. Having a classifier that can be analyzed in terms of which characteristics of trees lead to its predictions is therefore a desirable result.

The contents of this section, published in (Bringmann and Zimmermann 2005), are the result of joint work with Björn Bringmann at the Katholieke Universiteit Leuven.

The pattern language is again that of labeled rooted trees $\mathcal{L}_{tree}$, and we perform top-1 mining for class-correlating patterns. The stopping criterion takes the form $\sigma(p, \mathcal{D}) \geq \theta_{sig}$, which means that the score of the pattern mined has to match or exceed a significance threshold. A resulting tree set could look like the example given in Figure 6.1. Unlabeled instances are sorted down the tree

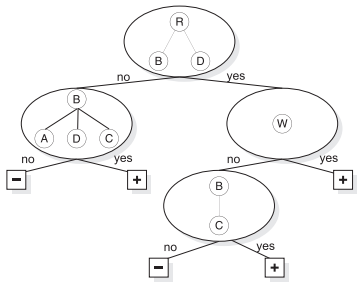


Figure 6.1: A tree set as produced by the TREE^2 algorithm

set according to which patterns they match until they arrive in a leaf and are then labeled with the majority class of training instances in that leaf.

We will explore how effective parallel mining is in mining compact sets of pattern that are still effective in modeling the underlying data. For the experimental evaluation, we therefore compared our approach to XRULES, and CTC on the XML data used in Zaki *et al.*'s publication (Zaki and Aggarwal 2003). The aim of this comparison lies in evaluating the effect that the iterative approach has when compared to post-processing techniques. Additionally, by treating patterns as binary features and learning decision trees, it's possible to build tree sets from a set of previously mined local patterns in a post-processing step. We compare TREE^2 to such an approach as well to test the assumption that manipulating the data before iterating local pattern mining leads to more useful patterns.

Furthermore we compare TREE^2 to techniques using different pattern languages and mining approaches on the regression-friendly subset of the Mutagenesis dataset, one of the most-used structured data sets in machine learning.

6.2.1 Experiment 1: Different techniques on XML data

In the first experimental evaluation, we test the assumption that parallel pattern mining leads to more compact pattern sets while retaining good accuracies. More concretely:

Q6.5 Does iterative tree mining lead to smaller pattern sets than exhaustive constraint satisfaction mining?

Q6.6 Does iterative tree mining lead to smaller pattern sets than post-processing exhaustive constraint satisfaction mining?

Q6.7 Do TREE^2's models well in terms of predictive accuracy?

In each setting we used one set of data for training and another one for testing. Following Zaki's notation, CSLOGx-y means that we trained on set x and tested on set y. XRULES and CTC are used with the same parameters as in Section 4.1. We evaluate different parameter settings for TREE^2:

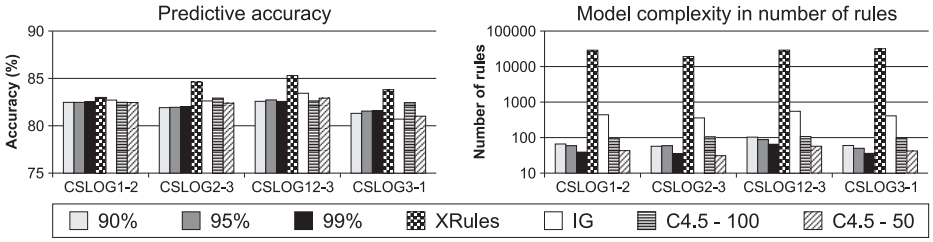


Figure 6.2: Accuracies and size in rules of the different approaches

- σ is *information gain* (IG), and χ^2, respectively
- θ_{sig} is set to 0.001 and 0.01 for IG
- θ_{sig} is set to the p-value for 90%, 95% and 99%

For the post-processing approach we mined we mined $Th_{50}(\mathcal{L}_{tree}, \mathcal{D}, \chi^2)$ and $Th_{100}(\mathcal{L}_{tree}, \mathcal{D}, \chi^2)$, used the respective pattern sets to transform the data into bitstring instances according to the found patterns. Decision tree are induced using C4.5 (Quinlan 1993) in the form of WEKA's J48 implementation (Frank and Witten 1999) with default parameters, referring to the two top-k settings as *C4.5 - 100* and *C4.5 - 50*, respectively. We compare the accuracies of the resulting classifiers against each other. We also report on the complexity of the model which we measure by the number of rules used by XRules, and CtC and by the number of leaves in the decision trees. This corresponds to the number of disjunctive rules that a tree can be transformed into, respectively.

Experimental results

Results are summarized in Figure 6.2. As can be seen, the accuracies of the induced classifiers do not vary much. The only approach that significantly outperforms (by 2-3%) the different decision tree techniques is XRules. At the same time, the size of XRules' models is also significantly larger. While the Tree^2 trees induced with *Information Gain* have several hundred leaves, and Tree^2 trees induced with χ^2 and C4.5 trees between 35 and 103 leaves, the smallest XRules model consists of more than 19000 rules. We can therefore state clearly that parallel mining leads to smaller pattern sets than exhaustive constraint satisfaction mining, answering Question 6.5.

Patterns tested in the inner nodes of trees built by Tree^2 consist of 2-6 edges only. Since this is similar to the size of patterns used in XRules' rules, complexity is really reduced and not just pushed inside the classifier. In comparing the other approaches, several things are noticeable. Raising the threshold from the 90% to the 95% significance level for χ^2-induced Tree^2 trees does not decrease accuracy (even improving it slightly in 3 cases). The tree size decreases, though, on average by 7.5 leaves from the 90% to the 95% setting.

Setting	CTC$_{MV}$	CTC$_{DL}$	CTC$_{AvgStr}$	CTC$_{WChi}$	TREE^2
CSLOG1-2	83.23	83.31	83.31	83.01	82.47
CSLOG2-3	83.95	83.90	83.92	82.83	81.91
CSLOG12-3	84.27	84.24	84.29	83.53	82.58
CSLOG3-1	83.50	83.77	83.63	83.53	81.31

Table 6.4: Predictive Accuracy for different classification strategies for CTC, and the TREE^2 classifier

Raising it further to the 99% level has no clear effect on predictive accuracy. Tree size is reduced further, however, decreasing by 18 leaves on average.

For the post-processing approach we mined patterns already correlating strongly with the classes and trained a classifier on them. This approach achieves competitive results regarding the accuracy. The clear drawback is that deciding on the number of features to use is not straightforward. Using only 50 instead of 100 features produces different outcomes. In some cases the accuracy does not change, in other cases the classifier using 50 features outperforms the one using 100 or vice versa. Also, the base-line approach using 100 patterns tends to use most of these, even if TREE^2 trees of similar quality are much smaller. This is an indicator that the features that are interesting with regard to the class distribution quickly degrade when they are evaluated on the changed distributions of subsets.

Additionally, using *Information Gain* as quality criterion shows mainly that it is difficult to make an informed decision on cut-off values. The accuracies and sizes shown refer to decision trees induced with $\theta_{sig} = 0.001$. For one thing, the resulting trees grow far bigger than the χ^2-trees. Additionally, the accuracies in comparison with the χ^2 approach vary, giving rise to one worse tree, one of equal quality and two better ones. None of the differences in accuracy is significant though. Inducing decision trees with $\theta_{sig} = 0.01$ lowers accuracy by 1.5 to 3 percentage points, with the induced trees still being larger than the χ^2 trees.

Finally, we are comparing TREE^2 trees to classifiers induced by CTC, a classifier that, as shown before, performs similarly to XRULES while already reducing the size of the classifier significantly. We reproduce the results for different classification methods from 4.1 in Table 6.4. We used the 90% p-value for χ^2 as θ_{sig} for both CTC and TREE^2.

As can be seen, CTC's larger, ordered sets generally perform slightly better than TREE^2. For the first setting, CSLOG1-2, the differences between the four variants of CTC, and TREE^2 are not significant at the 5% level. For the settings CSLOG2-3 and CSLOG12-3, the first three CTC variants (MV, DL, AvgStr) are significantly better than TREE^2 while CTC with the weighted χ^2 heuristic is not significantly better than TREE^2. Finally for the last setting, CSLOG3-1, all rule-based classifiers outperform TREE^2. Combined with accuracy comparison to XRULES, Question 6.3 has to be answered to the end that parallel mining trades off accuracy for cardinality of the pattern sets.

To put those results into perspective, the sizes of the different classifiers are

Setting	CTC	TREE^2	CTC$_{Val}$
CSLOG1-2	592	66	130
CSLOG2-3	497	57	150
CSLOG12-3	981	103	170
CSLOG3-1	546	60	220

Table 6.5: Size of the induced Models for CTC and TREE^2

compared with each other in Table 6.5. As can be seen, the size of the validation set-pruned CTC classifiers is still between 1.7 and 3.5 times that of TREE^2 trees, with the unpruned classifiers approaching roughly 9 times, answering Question 6.2. So even for the ordered sets we see a certain trade-off between size and classification accuracy, increasing model size to slightly increase accuracy, although to a lesser degree than what we observed for the comparison between TREE^2 and XRULES.

6.2.2 Experiment 2: Mutagenicity Data

For this setting, we chose the regression-friendly subset of the well known Mutagenicity dataset used in (Srinivasan et al. 1994). We compare with the results of the ILP system PROGOL reported in (Srinivasan et al. 1994, King et al. 1995) and the results of the base-line approach ((Bringmann and Karwath 2004)). In the base-line approach, instead of χ^2-significant patterns, a minimum frequency in one class (and maximum frequency in the other) is used and an unordered set mined. The patterns are then used as binary attributes and an SVM classifier trained on the resulting data representation.

The mutagenicity data set is represented as graph-structured data. Progol encodes patterns in first order logic while TREE^2 and the base-line work with tree-structured fragments. A transformation from the SMILES representation for molecules into so-called fragment-trees is used that is explained below.

First order languages can encode graph-structured patterns directly, and in the case of the FREQUENT SMILES approach, a powerful concept learner is applied in a post-processing step to tree patterns mined in a constraint satisfaction setting. Hence, the first question:

Q6.8 Does TREE^2 achieve comparable accuracy to the more sophisticated concept learners?

PROGOL additionally performs iterative pattern mining as well, using sequential covering. Therefore, the experimental evaluation is used to give evidence on:

Q6.9 How complex are the different models?

The SMILES Encoding

The Smiles language (Weininger 1988) is used by computational chemists as a compact encoding of molecular structure. Contrary to graph-formats which

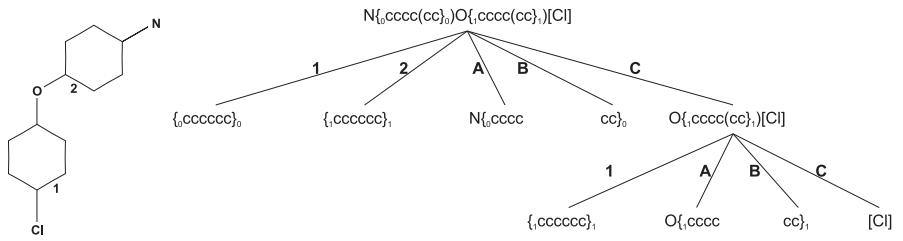


Figure 6.3: A molecule with the encoding $N-c1ccc(cc1)-O-c2ccc(cc2)-[Cl]$ and the corresponding fragment-tree

ask for explicit encoding of edges, it uses only ASCII characters to describe the molecule. Atoms are represented by their chemical symbols, parentheses are used to indicate branching points and numeric labels designate ring connection points. Hydrogen atoms are often omitted for the sake of compactness.

Many tools such as OpenBabel (The OpenBabel Software Community 2003) or Daylight (Daylight Chemical Information Systems,Inc. 2004) support SMILES. Using a decomposition-algorithm by (Karwath and De Raedt 2004), a Smiles-String can, after some reformatting, be decomposed into a so-called *fragment tree*. Since there is no *unique* Smiles-string for a molecule, the fragment tree is not unique either.

The decomposition-algorithm recursively splits the string into *cycles* $\{_xT\}_x$ and *branches* $A(B)C$. In the resulting fragment-tree the leaves contain pure cycles or linear fragments without further branches. The inner nodes of such a tree contain fragments still containing branches while the root node is the whole molecule. The edge labels denote the type of decomposition (i.e. the part of the branch or the number of the cycle). Thus, the leaves of a fragment-tree contain a lot of information decomposed into very small fragments. As in (Bringmann and Karwath 2004), we drop the edge labels and labeled all but the leaf nodes with a new, unique label. Hence, the tree-structure represents the abstract structure of the molecule with the chemical information in the leaves.

Figure 6.3 shows a molecule on the left-hand side which could be encoded by the SMILES-string

$$N-c1ccc(cc1)-O-c2ccc(cc2)-[Cl].$$

This string represents the same as

$$N\{_0cccc(cc\}_0)O\{_1cccc(cc\}_1)[Cl].$$

The corresponding fragment-tree is shown on the right-hand side of Figure 6.3.

Experimental results

We used ten-fold cross-validation to evaluate predictive accuracy for each approach. Reported are average accuracies and standard deviation (if known).

Table 6.6: Accuracies and complexity of the models on the mutagenicity dataset

Approach	Predictive Accuracy	Avg. Size of Model
TREE^2 χ^2	80.26±7.14	2.3 Nodes
TREE^2 *IG*	81.76±9	11.8 Nodes
PROGOL '94	80±3	9 Clauses
PROGOL '95	84	4 Clauses
FREQUENT SMILES	86.70	214 Patterns

For TREE^2, trees were induced at the 95% significance level for χ^2 and with a minimum significance constraint $\theta_{sig} = 0.01$ for *Information Gain*. The results reported in (Srinivasan et al. 1994) were achieved using PROGOL; in (King et al. 1995), numerical values suggested by experts were used as well. This work reports only an average accuracy. The resulting accuracies and the size of the corresponding theories are shown in Table 6.6.

As can be seen, for both measures TREE^2 gives similar results to the purely structural PROGOL approach, with the differences not significant. At the same time, the χ^2 induced model is far smaller than the other two. Again, the patterns tested against in the inner nodes are not overly complex (4-10 edges). When PROGOL uses the expert-identified attributes as well, its accuracy increases. Since we do not have access to the standard deviation of these experiments, we cannot make a significance statement. Finally, the base-line approach, which mined all patterns frequent in one class and not exceeding a given frequency in the other class, and built a model using these features in an SVM, significantly outperforms the TREE^2 classifiers. On the other hand, this is less of a statement about the quality of the pattern set and more about the capabilities of the SVM. Additionally, the amount of patterns used is larger than in the TREE^2 models by two orders of magnitude. Regarding Question 6.8, we can therefore state that TREE^2 induces classifiers of comparable accuracy. Additionally, TREE^2 when used led to small models, as did the first-order techniques. FREQUENT SMILES, on the other hand, uses a large set of patterns. This answers Question 6.9.

6.2.3 Conclusions

As our experiments show, parallel mining leads to the formation of compact yet accurate classifiers on structured data. The comparison to approaches in which large number of local patterns are mined, shows that the quality stays high but the understandability of the pattern set gets better, making such tree sets the arguably better pattern sets. We evaluated the mined patterns directly in the form of a decision tree but the use of more sophisticated classification strategies is possible. Parallel mining should also lead to smaller sets of good features for classification, as the results reported in (Fan et al. 2008) show.

All of these arguments suggest that parallel mining is a promising method for pattern set mining. Given our results from Section 6.1, the development of measures for unsupervised pattern mining, that is, without recourse to a class-label or target attribute, will make parallel mining even more useful.

6.3 Ensemble Trees – more complex local pattern formulations

So far in this chapter, we have discussed tree sets in which a single pattern is mined in each iteration, thus assembling a tree set. This pattern, however, governs how the data is manipulated for future iterations. An important insight in moving from top-1 mining towards sets of patterns in each iteration, therefore, lies in the inherent instability w.r.t. changes in the data that top-1 patterns display. We will illustrate this instability on the data below, assuming a classical binary decision tree, i.e. a decision tree where the pattern language takes the form $\mathcal{L}_{AV-pair} = \Sigma_{AV}$. This is a very restricted subset of the language $\mathcal{L}_{AV}$ we introduced in Definition 1.1.3. In classical decision trees, only a single test is included in each node, which corresponds to mining $Th_1(\mathcal{L}_{AV-pair}, \mathcal{D}, \textit{information gain})$.

index	A_1	A_2	...	class
1	v_2	v_1	...	+
2	v_1	v_2	...	−
3	v_2	v_1	...	+
4	v_1	v_1	...	+
5	v_1	v_2	...	−
6	v_1	v_2	...	−
7	v_1	v_1	...	−
8	v_2	v_1	...	+
...	...	...	...	...

Obviously, a change in either the value of A_1 in instance 4, or the value of A_2 in instance 7 would improve the strength of these attributes as a test, respectively. Similarly, removal of one of those two instances, for instance, as part of a cross-validation, changes which attribute is chosen. Since the choice of test affects how the data is split, this effect ripples down through the rest of the mining process. Assuming that A_1 and A_2 perform equally well on the rest of the data, the decision between them comes down to an arbitrary tie-breaker, giving the tree set a potentially rather different composition and performance.

There have been attempts to alleviate the problem in the context of decision tree induction, for instance, by using linear combinations of tests in nodes (Murthy 1997). The goal with using such more elaborate tests lies in stabilizing the trees and shrinking them. We suggest a simpler pattern language, $\mathcal{L}_{AV}$, and a different approach towards finding stable patterns, statistical quantification and small sets in each node.

We use the general parallel mining algorithm (Algorithm 13), introduced in Section 2.3.3, and perform top-k mining in each iteration, with k supplied by the user. Ideally this k should be small to facilitate a better understanding of the resulting tree set. Each node contains the entire (small) pattern set, which will be treated as an unordered set combined by majority voting, as we will explain below. Since a tree set is mined, however, the totality of patterns within a given branch has to be observed.

Within a node the order induced by top-k mining ensures totality, while between successive nodes this is guaranteed by the iterative process. Therefore, the resulting tree set satisfies this requirement. As a stopping criterion, we stop when it is either not possible to mine k statistically significant patterns, or when the patterns do not split off a large enough subset of the data anymore.

The aim of this is, of course, to reduce spuriousness. If not enough statistically significant patterns can be mined on the subset, statements about this subset start to become unreliable. In the same vein, when one of the subsets split off becomes too small, reliability issues start to appear and the mining should be stopped. A question to address here is how exactly this split is performed. We discuss this issue in the following section.

6.3.1 New notions of matching

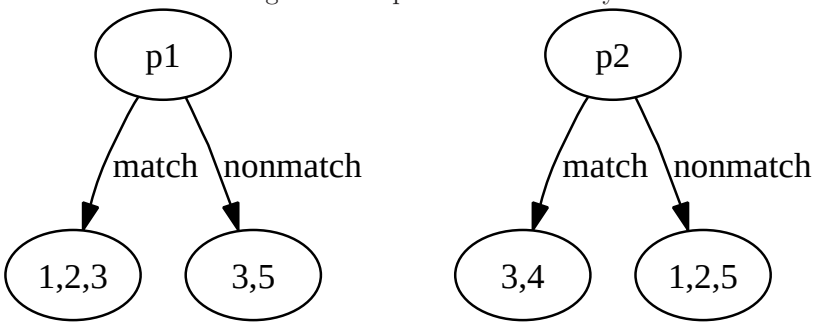
Once a set of several patterns is used in a node to split the data, the question naturally arises as to how to perform this split. We will illustrate both the potential pitfalls and the solutions we chose on the following set of instances:

index	A_1	A_2	A_3	A_4	A_5	class
1	v_1	v_1	v_2	v_1	v_1	$+$
2	v_1	v_1	v_1	v_1	v_1	$+$
3	v_1	v_2	v_1	v_1	v_2	$-$
4	v_1	v_1	v_1	v_2	v_2	$-$
5	v_2	v_1	v_2	v_1	v_1	$+$

Let us assume a single conjunctive pattern that is used to split the data, of the form: $A_1 = v_1 \wedge A_2 = v_1$. As usual, the splitting decision is simply based on the match of the pattern on the instances and we adopt the convention that matching instances are sorted down the left path, non-matching ones down the right. As we can see, instances 1, 2, and 4 will now be sorted to the left, with 3 and 5 going right (cf. Figure 6.4 (a)). This also hints at a side-effect of maximizing the information gain value of patterns: since IG rewards patterns that reduce entropy in the subsets, they usually also "predict" one of the two classes. This is, however, an implicit effect: $A_1 = v_1 \wedge A_2 = v_1 \Rightarrow +$

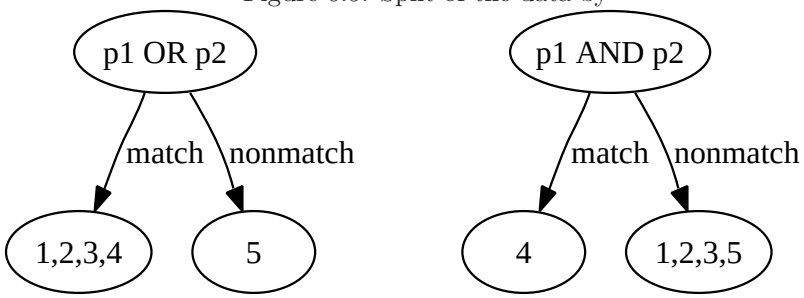
A second pattern $A_1 = v_1 \wedge A_5 = v_2 \Rightarrow -$ would sort instances 3 and 4 to the left, the others to the right (cf. Figure 6.4 (b)). If we combine the two patterns into a set and use the usual notion of pattern set matching (that at least one individual pattern has to match), the only instance that gets sorted to the right is instance 5, as seen in Figure 6.5 (a). This is somewhat unsatisfactory: each

Figure 6.4: Splits of the data by



(a) A pattern predicting positive class (b) A pattern predicting negative class

Figure 6.5: Split of the data by



(a) Two patterns combined by logical OR (b) Two patterns combined by logical AND

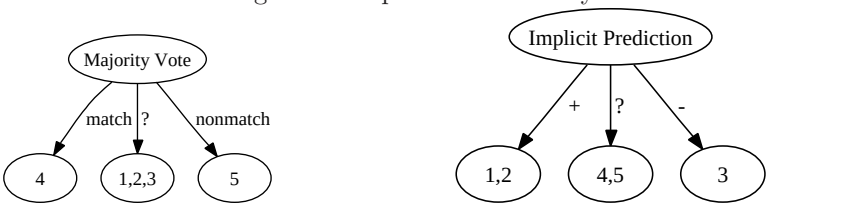
individual pattern separated the classes better than the set does, it does not really leverage the relation between patterns. An alternative lies in treating pattern sets like local patterns: each element has to match an instance.

If we use this notion, we have the dual problem that only one instance is split from the others, instance 4 (cf. Figure 6.5 (b)). While this is not a bad split, we can expect in each iteration only small subsets to be split off. The reasons is that this notion is rather restrictive in requiring *all* patterns to match. Instead, one can take inspiration from classification using unordered pattern sets and use a majority vote. In this particular case, as illustrated in Figure 6.6 (a), this does not improve things much because while there is agreement on instances 4 and 5, instances 1, 2, and 3 would be sorted in neither or an arbitrary direction.

To solve this final obstacle, on the one hand we enforce that there will always be an odd number of patterns in a node. Additionally, given a pattern $p \Rightarrow +$ ($p \Rightarrow -$), we assume an implicit prediction $\neg p \Rightarrow -$ ($\neg p \Rightarrow +$). In other words, whenever a pattern does not match an instance, we treat this as a vote for the class this pattern *does* not predict.

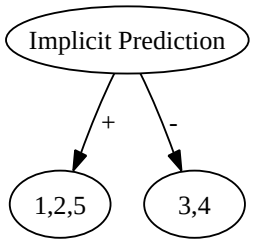
Already for two patterns the situation is improved, with instances one, two predicted positive (and sorted left), and instance three predicted negative. If

Figure 6.6: Split of the data by



(a) Two patterns combined by Majority Voting (b) Two patterns, exploiting their implicit predictions as well

Figure 6.7: Split of the data by three patterns, exploiting their implicit predictions



we add a third pattern to break the tie regarding instances four and five, such as $A_1 = v_1 \wedge A_5 = v_2 \Rightarrow -$, we have a flexible matching scheme which takes the semantics of patterns into account as shown in Figure 6.7. This should lead to relatively balanced splits that separate classes well, just what *IG* rewards.

We call such a tree set an *ensemble tree*, since the set of (implicitly) predictive patterns that is resides at each node, and is combined via majority voting, have similarities with ensembles of rules. *Ensemble-Trees* are different from the type of tree sets we discussed so far in that both the usual notion of matching for local patterns and for entire pattern sets have been modified.

6.3.2 Experimental evaluation

Ensemble-Trees are tree sets consisting of conjunctive patterns. In this particular setting, concept learning, however, different solutions involving tree sets of one kind or the other already exist. First of all, there are classical decision trees, as embodied by C4.5, which use single patterns expressed in $\mathcal{L}_{AV}$ in the set. Obviously, *Ensemble-Trees* are more complex than decision trees, so there are two questions that arise from this:

Q6.10 Does this increased complexity translate into better classification performance, especially in the absence of post-pruning?

Q6.11 Are the resulting *Ensemble-Trees* more stable with regard to changes in the data (a prime motivation we gave above)?

Q6.12 Does the higher complexity of patterns in the nodes of *Ensemble-Trees* translate to fewer nodes per tree, compared to decision trees?

Decision trees can also be understood as local patterns themselves, and there have been attempts to build sets of decision trees to reduce the volatility of predictions stemming from the instability of decision trees. Two of the best-known techniques in this regard are *Boosting*, for instance, exemplified by the AdaBoost algorithm, and *Bagging* (which was developed specifically with decision trees in mind). *Bagging* builds unordered sets of trees, as we will discuss in more detail in Section 6.3.3. AdaBoost (and Boosting in general) builds ordered sets of trees in a manner very much like sequential re-weighting. Those techniques are usually referred to as *ensemble methods* since in addition to the sets themselves, there are classification methods inherent in them. Similarly to the use of other local pattern sets we have seen before, *Bagging* uses majority voting, and *Bagging* a weighted voting strategy.

The two questions arising in this context are

Q6.13 Do tree sets perform better than sets of trees when it comes to classification accuracy?

Q6.14 Have tree sets lower complexity than sets of trees?

Experimental setup

We use several UCI data sets to compare *Ensemble-Trees* to C4.5, Bagging, and AdaBoost, each in their WEKA (Frank and Witten 1999) implementations with C4.5 as the weak classifier.

Since we limit ourselves to nominal attribute values in this work, numerical attributes were discretized, using ten bins of equal width. With the branch-and-bound technique used for inducing the rule ensembles being limited to binary classes, we used only data sets with binary class labels. However, given that every classification problem can be translated into a number of *one-against-one*, or *one-against-all* problems, this is not a significant drawback of our approach.

We performed experiments with the minimum leaf size parameter m set to $2, 3, 4, 5, 10$ and in case of large data sets, 10% of the training data. As the interestingness measure σ, information gain was used. Bagging and AdaBoost were set to 10 iterations of inducing decision trees, and we built *Ensemble-Trees* with $k = 3$, and $k = 5$, respectively. AdaBoost was used both in the resampling and the re-weighting mode. Decision tree and ensemble method results are reported on pruned trees while *Ensemble-Trees* are always unpruned.

Experiment 1: Predictive accuracy

Predictive accuracy was evaluated using a stratified 10-fold cross-validation. Table 6.7 reports mean and standard deviation for a minimum leaf size $m = 5$, except for the Trains data set, where $m = 2$. While details vary, the main trends can be observed for all minimum leaf sizes evaluated in the experiments.

Table 6.7: Predictive accuracies for decision trees/*Ensemble-Trees* with minimum leaf size of 5

Data set	C4.5	$ET_{k=3}$	$ET_{k=5}$	Bagging	AdaBoost$_{RS}$	AdaBoost$_{RW}$
Breast-Cancer	$73.42 \pm 5.44\bullet$	78.69 ± 4.34	80.14 ± 6.16	73.77 ± 6.98	$67.09 \pm 10.10\bullet$	$66.77 \pm 6.81\bullet$
Breast-Wisconsin	94.56 ± 2.93	95.28 ± 1.35	95.14 ± 1.80	95.42 ± 2.76	96.28 ± 1.81	95.99 ± 1.14
Credit-A	84.20 ± 2.93	85.51 ± 2.56	85.51 ± 2.56	85.22 ± 2.35	82.75 ± 3.45	$82.03 \pm 4.44\bullet$
Credit-G	$71.90 \pm 3.96\bullet$	80.33 ± 2.00	79.10 ± 5.09	$74.40 \pm 4.06\bullet$	$72.60 \pm 3.24\bullet$	$70.30 \pm 4.00\bullet$
Heart-Statlog	82.96 ± 8.04	81.85 ± 5.08	79.63 ± 6.82	80.74 ± 8.52	80.37 ± 7.82	78.52 ± 7.57
Hepatitis	84.50 ± 6.22	89.58 ± 5.61	90.92 ± 5.54	$83.25 \pm 5.35\bullet$	$83.17 \pm 7.07\bullet$	$82.71 \pm 8.27\bullet$
Ionosphere	88.60 ± 5.88	91.44 ± 3.82	88.92 ± 5.80	91.15 ± 4.37	90.87 ± 5.69	91.15 ± 6.25
Molec. Biol. Prom.	78.09 ± 14.57	83.73 ± 9.28	83.64 ± 11.36	88.00 ± 13.00	85.73 ± 10.96	88.64 ± 5.94
Mushroom	$100 \pm 0\circ$	99.95 ± 0.06	99.95 ± 0.06	96.31 ± 5.94	$100 \pm 0\circ$	$100 \pm 0\circ$
Tic-Tac-Toe	$91.76 \pm 3.81\circ$	76.20 ± 1.47	72.86 ± 4.59	$96.87 \pm 1.55\circ$	$98.02 \pm 1.59\circ$	$96.97 \pm 2.62\circ$
Trains	60.00 ± 51.64	50.00 ± 52.70	50.00 ± 52.70	40.00 ± 51.64	80.00 ± 42.16	50.00 ± 52.70
Voting-Record	$95.85 \pm 2.83\bullet$	98.39 ± 1.11	98.62 ± 1.19	$95.62 \pm 2.29\bullet$	$94.94 \pm 3.04\bullet$	$96.08 \pm 3.09\bullet$

The table shows that *Ensemble-Trees* perform mostly well w.r.t. classification. In several cases, *Ensemble-Trees* are significantly better than ensemble methods ($\bullet$ denotes a significant loss of a technique at the 5%-level), while being outperformed only on Mushroom (barely), and Tic-Tac-Toe ($\circ$ denotes a significant win at the 5%-level). This points once again to the phenomena we have experienced before: Tree sets that allow the refinement of descriptions for subsets have a better chance of catching phenomena than sets whose members aim to accurately describe a subset of the data on their own. *Ensemble-Trees* also perform worse than decision trees only once (on Tic-Tac-Toe again), indicating that the increased complexity pays off. Question 6.10 and 6.13 can be answered positively: increasing the complexity of the local pattern mining step leads to tree sets that are better suited to classification.

Inspection of the standard deviations, however, gives a first indication that *Ensemble-Trees* do not turn out to be more stable performance-wise than decision trees when the data composition changes. In fact, *Ensemble-Trees* using only three rules per node show a *smaller* standard deviation, corresponding to a smaller variance, with regard to accuracy accuracy than *Ensemble-Trees* with sets of patterns in each node. We will investigate this issue further in the next section

Experiment 2: Size and stability of *Ensemble-Trees* in comparison to decision trees

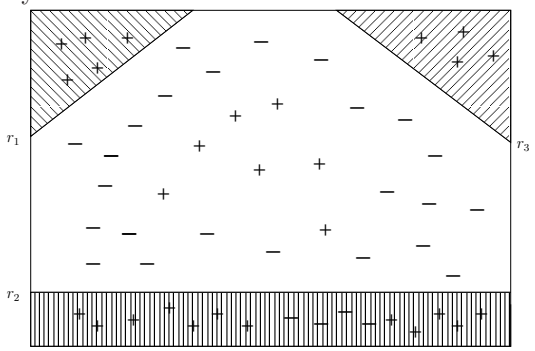
The second question we evaluated was concerned with stability of induced trees with regard to changes in the data. We used the 10-fold cross-validation mechanism already employed in the accuracy estimation to simulate changes in the underlying data, and report on size characteristics of the trees in Tables 6.8 and 6.9. For both C4.5 decision trees and *Ensemble-Trees*, we report the mean and standard deviation of sizes (number of nodes) and maximal depths, i. e. length of the longest branch, of the different trees.

Ensemble-Trees typically have somewhat fewer nodes than classical C4.5 decision trees, due to the more expressive tests in the nodes. The cases where this trend does not hold (the Hepatitis, and Voting-Record data sets) or is ex-

Table 6.8: Number of nodes per tree for C4.5 and *Ensemble-Trees*, respectively

Data set	C4.5	$ET_{k=3}$	$ET_{k=5}$
Breast-Cancer	13.6 ± 6.4	7.6 ± 3.5	9.4 ± 5.1
Breast-Wisconsin	14.8 ± 1.5	13.4 ± 2.8	9.0 ± 2.1
Credit-A	19.2 ± 4.2	16.4 ± 6.2	10.6 ± 6.1
Credit-G	86.4 ± 9.2	22.8 ± 6.1	18.8 ± 8.9
Heart-Statlog	14.4 ± 2.3	11.2 ± 2.2	10.8 ± 3.9
Hepatitis	6.2 ± 3.0	10.6 ± 4.1	9.2 ± 3.0
Ionosphere	17.2 ± 3.7	12.2 ± 6.0	10.4 ± 4.0
Molec. Biol. Prom.	11.6 ± 1.0	5.4 ± 0.8	6.4 ± 1.3
Mushroom	17	17	21
Tic-Tac-Toe	53.4 ± 2.8	3	2.8 ± 0.6
Trains	3	3	3
Voting-Record	8.2 ± 1.0	13.0 ± 3.4	13.2 ± 4.0

Figure 6.8: Binary class data and decision surfaces of three discriminatory rules



aggerated (the Breast-Cancer, and Credit-G data sets) are also the ones where *Ensemble-Trees* perform best compared to the other approaches. However, in the case of the Tic-Tac-Toe data set, the small number of nodes is actually a symptom of an underlying characteristic, with another being the severe underperformance of *Ensemble-Trees*, compared to the other methods.

To explain the mechanism at work here, Figure 6.8 shows binary class data points in two-dimensional space, and the decision surfaces of three rules r_1, r_2, r_3. The subsets of the data covered by each rule are shown shaded. As can be seen, each of these rules predicts the positive class, advocating that the instances they cover be sorted to the left. Additionally, however, all of them advocate the sorting of the instances covered by the other two rules to the right. The result of the majority vote on this is that all instances are sorted to the right, the left subset is empty and thus its size less than m, leading to the formation of a leaf. The rather aggressive pre-pruning effected by this stopping criterion becomes problematic on data sets with small, non-overlapping sub-regions in the data. Tic-Tac-Toe is such a data set, as indicated by the rather large number of

Table 6.9: Averaged maximal depths for C4.5 trees and *Ensemble-Trees*, respectively

Data set	C4.5	$ET_{k=3}$	$ET_{k=5}$
Breast-Cancer	4.1 ± 1.5	2.90 ± 1.2	3.5 ± 2.2
Breast-Wisconsin	4.4 ± 0.5	4.2 ± 0.9	3.9 ± 0.9
Credit-A	6.7 ± 1.3	5.8 ± 2.7	4.0 ± 2.7
Credit-G	23.9 ± 2.3	7.9 ± 2.1	18.8 ± 8.9
Heart-Statlog	3.2 ± 0.8	3.8 ± 0.8	3.6 ± 1.0
Hepatitis	1.1 ± 1.5	4.7 ± 2.0	4.0 ± 1.3
Ionosphere	6.9 ± 1.6	5.6 ± 3.0	4.6 ± 1.8
Molec. Biol. Prom.	2.8 ± 1.0	2.2 ± 0.4	2.7 ± 0.7
Mushroom	4	5	5
Tic-Tac-Toe	6.0	1	0.9 ± 0.3
Trains	1	1	1
Voting-Record	2.6 ± 0.5	4.2 ± 0.8	4.4 ± 1.1

nodes (and therefore leaves) in – pruned – C4.5 decision trees. So in answering Question 6.11, we can state that there is a certain decrease in the number of nodes per tree set, due to the increased complexity of patterns, but not in all cases.

The expected stabilization of trees regarding changes in the data, however, cannot be observed. Neither in the number of nodes, nor in the maximal depths of trees do *Ensemble-Trees* markedly decrease the standard deviation over folds, compared to C4.5 pruned trees. Quite contrary, while the trees are shallower, and mostly smaller, *Ensemble-Trees* often show greater variance in both characteristics than classical decision trees do.

Since the size measurements were extracted after post-pruning, we finally compare unpruned C4.5 trees to *Ensemble-Trees* in an attempt to understand how much of a stabilization effect post-pruning provides to the decision trees. The relevant numbers are shown in Tables 6.10 and 6.11, with the characteristics of *Ensemble-Trees* duplicated from Tables 6.8 and 6.9.

While Table 6.10 suggests that the reduction in variance for the size of a decision tree is a result of the post-pruning operation, and Table 6.11 shows similar effects regarding maximal depths of the tree, this still means that *Ensemble-Trees* are structurally not more stable regarding changes in the data than classical decision trees. While the use of sets of conjunctive patterns as tests could bring more stability to the final tree, this promise remains unfulfilled, answering Question 6.12. A potential improvement could lie in a better solution for the combination strategy.

Experimen 3: Comparison of the size of tree sets, ordered, and unordered sets of trees

The final question to be addressed is concerned with whether tree sets of conjunctive patterns are smaller than ordered and unordered sets of trees. Since

Table 6.10: Number of nodes per tree for unpruned C4.5/*Ensemble-Trees*

Data set	C4.5	$ET_{k=3}$	$ET_{k=5}$
Breast-Cancer	41.0 ± 8.7	7.6 ± 3.5	9.4 ± 5.1
Breast-Wisconsin	19.4 ± 4.9	13.4 ± 2.8	9.0 ± 2.1
Credit-A	35.8 ± 11.2	16.4 ± 6.2	10.6 ± 6.1
Credit-G	147.0 ± 10.1	22.8 ± 6.1	18.8 ± 8.9
Heart-Statlog	26.8 ± 3.3	11.2 ± 2.2	10.8 ± 3.9
Hepatitis	24.8 ± 2.6	10.6 ± 4.1	9.2 ± 3.0
Ionosphere	17.2 ± 3.7	12.2 ± 6.0	10.4 ± 4.0
Molec. Biol. Prom.	11.6 ± 1.0	5.4 ± 0.8	6.4 ± 1.3
Mushroom	21	17	21
Tic-Tac-Toe	68.2 ± 7.3	3	2.8 ± 0.6
Trains	3	3	3
Voting-Record	8.8 ± 1.1	13.0 ± 3.4	13.2 ± 4.0

Table 6.11: Averaged maximal depths for unpruned C4.5 trees/*Ensemble-Trees*

Data set	C4.5	$ET_{k=3}$	$ET_{k=5}$
Breast-Cancer	9.5 ± 2.4	2.9 ± 1.2	3.5 ± 2.2
Breast-Wisconsin	5.7 ± 1.1	4.2 ± 0.9	3.9 ± 0.9
Credit-A	11.3 ± 2.3	5.9 ± 2.7	4.0 ± 2.7
Credit-G	26.1 ± 2.3	7.9 ± 2.1	18.8 ± 8.9
Heart-Statlog	5.2 ± 1.0	3.8 ± 0.8	3.6 ± 1.0
Hepatitis	4.1 ± 0.9	4.7 ± 2.0	4.0 ± 1.3
Ionosphere	10.4 ± 1.2	5.6 ± 3.0	4.6 ± 1.8
Molec. Biol. Prom.	2.8 ± 1.0	2.2 ± 0.4	2.7 ± 0.7
Mushroom	6	5	5
Tic-Tac-Toe	6.6 ± 0.7	1	0.9 ± 0.3
Trains	1	1	1
Voting-Record	2.9 ± 0.6	4.2 ± 0.8	4.4 ± 1.1

Table 6.12: Number of attribute-value pairs per tree for *Ensemble-Trees*, accumulated number of nodes for all trees of the respective ensembles

Data set	$ET_{k=3}$	$ET_{k=5}$	Bagging	AdaBoost$_{RS}$	AdaBoost$_{RW}$
Breast-Cancer	23.7 ± 14.4	55.4 ± 38.3	425.4 ± 19.4	485.2 ± 18.2	509.4 ± 14.7
Breast-Wisconsin	29.7 ± 6.1	36.6 ± 8.2	154.2 ± 12.2	334.4 ± 12.2	331.8 ± 15.4
Credit-A	58.2 ± 19.7	60.5 ± 48.2	226.6 ± 8.5	698.6 ± 17.0	725.2 ± 30.4
Credit-G	112.4 ± 26.6	171.2 ± 85.7	896.0 ± 23.7	1212.2 ± 25.7	1239.2 ± 27.9
Heart-Statlog	34.3 ± 9.4	57.7 ± 24.4	193.2 ± 14.5	302.6 ± 14.3	310.0 ± 14.5
Hepatitis	44.9 ± 18.5	67.8 ± 27.5	86.6 ± 10.4	153.6 ± 7.9	165.4 ± 7.9
Ionosphere	27.1 ± 17.9	37.6 ± 18.4	165.6 ± 7.6	243.4 ± 8.9	253.0 ± 9.2
Molec. Biol. Prom.	13.8 ± 3.0	30.0 ± 8.8	98.2 ± 8.3	83.4 ± 3.9	85.8 ± 6.0
Mushroom	51.0 ± 0.0	107.7 ± 1.2	153.8 ± 5.6	17	169.4 ± 15.0
Tic-Tac-Toe	7.0 ± 0.0	10.1 ± 3.6	573.4 ± 9.2	700.8 ± 31.4	736.6 ± 23.7
Trains	5.4 ± 1.6	10.8 ± 2.3	30.2 ± 0.6	23.5 ± 7.5	28.4 ± 0.8
Voting-Record	36.7 ± 11.3	70.0 ± 27.4	75.6 ± 6.7	184.2 ± 18.0	181.6 ± 15.7

the ensemble methods induce a different tree in each iteration, averaging over the iterations *and* the folds becomes a difficult endeavor, and we report on the accumulated number of nodes (among all trees per fold). As we wrote above, we set the number of iterations to 10 for the ensemble methods. Bagging will always use all 10 iterations while Boosting might stop earlier if the performance of the induced trees suffices (again similar to re-weighting). To make the comparison of complexities explicit, we do not list the number of nodes for *Ensemble-Trees* but instead the number of conjuncts (attribute-value pairs) involved in patterns in test nodes, averaged over the 10 folds.

Inspection of Table 6.12 shows that *Ensemble-Trees* always have far fewer tests than ensembles of trees have nodes (and therefore tests), answering Question 6.14 positively. The only exception is AdaBoost with resampling on the Mushroom data set. This is once again consistent with the results we have seen in Sections 6.1 and 6.2, with tree sets typically being of smaller cardinality than (un)ordered sets. Given that the ensemble methods do not improve markedly on the accuracy of *Ensemble-Trees* in most cases, quite a few more bags/iterations would be needed for better performance (except for the Mushroom data set), in turn leading to far larger sets of trees. *Ensemble-Trees* therefore reduce complexity in comparison to sets of trees.

6.3.3 Related approaches

As we stated above, Bagging and Boosting can be considered techniques for inducing ordered and unordered sets of trees, respectively. Specifically Boosting shares characteristics with the sequential re-weighting technique we discussed earlier in this part, in Section 5.2. In both approaches, voting strategies are used to combine decision tree predictions.

One of the best known ensemble techniques, Bagging (Breiman 1996), was proposed specifically with decision trees as *weak learners* (local patterns) in mind. Each tree is induced on a boot-strapped sample (created via sampling with replacement) of the data, forming an unordered set of trees, shown in

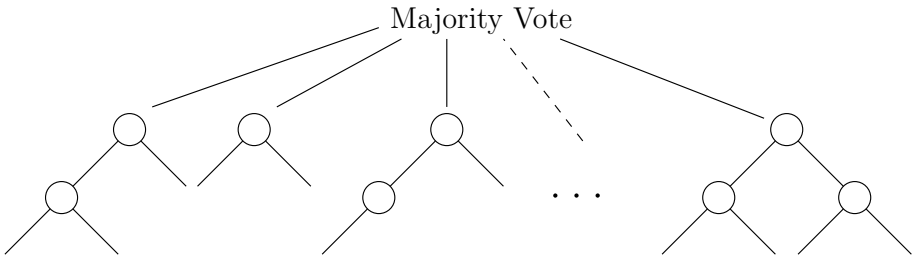


Figure 6.9: Set of trees created by Bagging

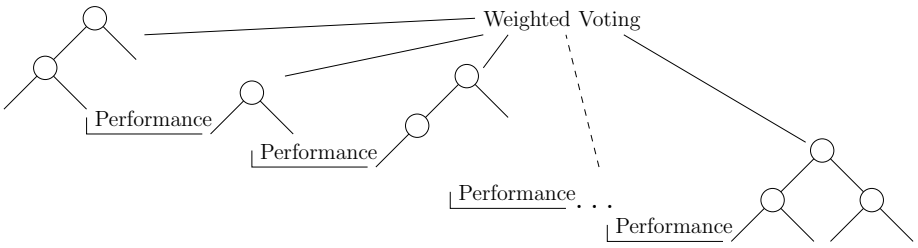


Figure 6.10: Ordered set of trees created by Boosting

Figure 6.9. When unseen data is encountered, majority voting of the trees' predictions is performed to form the final prediction. The random resampling of the data ensures that each tree will truly describe only local phenomena and have no feedback from other patterns' performance apart from the voting phase. The aim is that each pattern's peculiarities are cancelled out by the influence of other patterns during that phase. In a refinement of his technique, Breiman *et al.* proposed RANDOM FORESTS (Breiman 2001), in which not only the training data but also the tests from which to choose for a node are randomly selected, creating very powerful classifiers. They also highlight the main problem with this kind of approach: each local pattern becomes almost meaningless in a global context and the full set is hard if not impossible to interpret.

A more principled way of changing distributions in the data for the purpose of inducing patterns that reflect other patterns' effect is embodied by the Boosting approach, as in the well known ADABOOST system (Freund and Schapire 1997). Again, decision trees act as local patterns but are induced incrementally with each tree being evaluated on the data and misclassified instances being given more importance by resampling or re-weighting. This is a similar approach to the weighted covering technique we discussed in 5.2, and the technique results in an ordered set in the sense that each pattern is influenced by its predecessor's performance (for which the same holds), as shown in Figure 6.10. A significant difference to ordered sets of rules for classification lies in that a weighted voting mechanism is used for the final classification (since re-weighting is not based on matching but on performance). Different Boosting algorithms

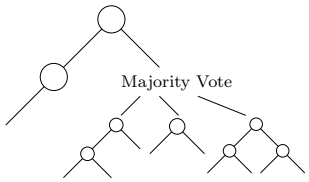


Figure 6.11: Option tree - a tree set of tree sets

use different ways of weighting data and hypotheses, based on results from computation learning theory. Based on these results, they can be proved to allow a set of weak learners to approach a strong learner's performance to arbitrary degree but for that a considerable (and a priori unknown) number of iterations has to be performed. This aspect is also the main difference to the re-weighting techniques we have discussed so far (Lavrač et al. 2004, Yin and Han 2003) since the metrics for re-weighting are chosen somewhat arbitrarily in those settings.

An early approach towards tree sets of complex patterns are the so-called *option-trees* (Kohavi and Kunz 1997). In option nodes (whose number and level in which they may appear can be controlled by the user), *all* tests are kept,whose quality are within a certain range of each other. The data on which they were evaluated is copied to *each* of them and further parallel covering performed below them. What this in effect leads to, even though the authors did not frame it in this way, is a tree set which in non-option nodes has a top-1 $\mathcal{L}_{AV}$ pattern. In option nodes, however, a set of tree sets is assembled, whose inner nodes might be option nodes themselves. The (extremely simplified) structure of such a set is shown in Figure 6.11, a "true" $Tree^2$ (or $Tree^n$ actually since the number of internal trees is unknown). Unseen data is copied in the same way as training data was, with the predictions of each option node combined via majority voting.

6.3.4 Conclusions

Extending tree set mining further by not only increasing the complexity of individual patterns but also the number of patterns mined in each iteration has its benefits and its drawbacks. The first benefit lies in reduced complexity, in terms of number of iterations and number of attribute-value pairs, compared to sets of trees. The second benefit lies in improvements in classification accuracy in several cases, compared to sets of trees and regular decision trees.

That this improvement in classification accuracy does not occur in all cases, and that it is connected to the way the patterns of each iteration are combined, indicates the drawback of this model. Contrary to tree sets using individual patterns, *Ensemble-Trees* need to employ a more sophisticated matching operator, whose design has effects on the usefulness of the full pattern set. There is therefore a need to improve the matching and strategies, and in that way the mining, of *Ensemble-Trees* to make tree sets more flexible and more useful for

KDD.

6.4 Summary and Future Work Directions

The second chapter on iterative mining was concerned with tree sets and how to mine them using parallel mining processes. Similarly to sequential mining, the data is manipulated after the local pattern mining step of each iteration. The main difference, however, lies in the fact that while sequential mining implicitly assumed that an iteration gave a relatively complete description of the data covered, this assumption is absent in parallel mining. The assumption is rather that such a description is coarse or general or high-level and that more fine-grained, specific or lower-level phenomena can occur. Therefore, instead of discounting or even removing data for the purpose of deriving future patterns, data is split into subsets that correspond to the (non)coverage of mined patterns. Should the description actually be as complete as sequential mining assumes, this will automatically stop the pattern set mining process (given suitable stopping criteria).

Removing the burden of complete description has the side-effect that parallel mining often leads to smaller pattern sets since weaknesses of a description can be corrected easily by the more fine-grained patterns mined later. Additionally, the strict order that links patterns in ordered sets together can be relaxed: patterns can actually occur in the set without having a particular relation.

As in the chapter on sequential mining, we explored the opportunities that are created the decomposition of parallel mining into its local pattern mining and pattern set mining components. We demonstrated the usefulness and properties of parallel pattern set mining in three experiments:

In Section 6.1, we discussed how to use parallel mining to induce a dendrogram composed of conjunctive descriptions of clusters. While parallel mining for cluster formation has been discussed before, this is the first time that *conceptual clustering* descriptions that go beyond single tests are induced. In the comparison with existing clustering solutions, we showed that this method induces high-quality clusters that are described concisely by the induced descriptions. Additionally, clustering by pattern set mining gives the user much control over the process, leading the interpretable solutions.

In Section 6.2, parallel mining was performed on tree-structured data in the context of concept learning. Pattern set mining on structured data via parallel mining is another topic that has been given little attention so far, even though the approach is effective. The introduced algorithm, called Tree^2 gives rise to far smaller sets of patterns than post-processing methods or sequential mining, even if the pattern language of the latter is more expressive. Particularly in the case of parallel set mining as a post-processing technique for a set of local patterns that has been mined before, iterative mining produces smaller sets since it implicitly takes preceding patterns into account during mining.

Finally, in Section 6.3, we upgraded the parallel pattern set mining setting by mining more than one pattern in each iteration, moving further away from the

classical decision tree induction setting. The context was again that of concept learning and we illustrated some of the issues that arise from mining several patterns per iteration and suggested ways of treating them. In the comparison to sets of trees and to classical decision trees, we could show improvements in performance. Additionally, especially in comparison with the sets of trees (in which decision trees form the local patterns), the overall size of the set (and therefore interpretability) was reduced strongly.

As those evaluations show, parallel pattern set mining is a strong mechanism for inducing relatively small, effective tree sets of patterns for a variety of tasks. Given the greater flexibility of tree sets compared to ordered sets (which they can represent but not vice versa), the approach is potentially better-suited to iterative mining. The trade-off is very likely to have a higher computational cost, however, since data are not necessarily removed from further consideration. The multi-target top-k local pattern mining technique employed in Section 6.1 can be used to mine for an arbitrary number of target attributes. Therefore, it is possible to induce tree sets consisting of complex patterns in the context of, for instance, multi-target predictive trees as the ones introduced in (Vens et al. 2008) as well.

Conclusion of Part III

This part was concerned with pattern set mining techniques that originate from machine learning and proceed in an iterative fashion. In contrast to the approaches discussed in Part II, pattern set mining is not a single step that follows local pattern mining but instead data is manipulated based on the local patterns mined, and mining re-iterated on the modified data.

The two paradigms existing in machine learning are sequential and parallel mining. Chapter 2.3.1 considers sequential mining, in which data that is covered (matched) by the local pattern or patterns mined is discounted in later iterations. Sequential covering achieves this discounting by removing covered data from further consideration while sequential re-weighting decreases the weight of covered instances.

While both of those techniques are usually discussed in combination with specific heuristic local pattern mining components in the machine learning literature, they can be discussed in connection with different, exhaustive techniques. We evaluated the difference between the use of a heuristic and an exhaustive local pattern mining step in Section 5.1. Each of the two choices has a potential disadvantage: heuristic local pattern mining conceivably leads to suboptimal solutions since the patterns set mining component also proceeds heuristically. Exhaustive mining, on the other hand, is potentially much more computationally expensive. Both disadvantages do not materialize, however, as our experiments show. Sequential mining directs heuristic mining in such a way that the resulting classifiers perform equally well. Heuristic local pattern mining, on the other hand, is *not* more efficient than exhaustive mining due to the fact that the latter typically assembles smaller pattern sets. This is a major new insight of our work.

Parallel mining does not decrease the importance of covered data for future iterations but instead divides the data into covered and uncovered subsets. Since there is less pressure on patterns to describe data accurately, we expected this technique to lead to pattern sets of lower cardinality. This technique, which originates in decision tree induction, is mainly applied in concept learning, on attribute-value data, and with pattern languages of relatively low complexity. In Chapter 2.3.3, we evaluated the effects of changing these characteristics. In Section 6.1, we increased the complexity of the language, mining conjunctive patterns in each iteration, which becomes possible by using exhaustive mining with upper-bound pruning. By applying the new system, CG-Clus, in the

context of *conceptual clustering*, we showed that it induces high-quality clusters that have compact descriptions. We used the second section of Chapter 2.3.3 to explore the direct mining of tree-structured patterns for decision trees. This is a deviation from the usual methods in which either large sets of predictive patterns are mined or data re-encoded and decision trees induced on the new presentation. The resulting classifiers were once again very compact but paid for this with a slight decrease in accuracy compared to a large unordered set of patterns that uses a weighted voting scheme. Finally, in Section 6.3, we increased the number of patterns mined in each iteration, in addition to mining conjunctive patterns. We discussed issues arising from this decision, such as the interpretation of the "match" of a pattern set, and evaluated the resulting classifiers empirically. The experiments showed increased accuracy and lower complexity than sets of trees used for classification.

The techniques we discussed in this part were limited to the sequential and parallel mining paradigm and we explored how the abstraction from monolithic system allows to use different approaches to the local pattern mining step. In general, however, iterative mining does not have to proceed in this way. An example of an iterative mining technique that is performed differently, was proposed by Rückert *et al.* in (Rückert and Kramer 2007). Using a stochastic random walk, the authors propose mining class-correlating graph patterns, and iterating this process, maximizing a measure trading off class-correlation and similarity to patterns mined before. Those patterns are then shown experimentally to be more suitable for classification tasks, leading to smaller feature sets, better performance or both.

In a slightly different vein are the nFoil and kFoil approaches by Landwehr *et al.* (Landwehr et al. 2005, Landwehr et al. 2006). The goal of these works is the induction of clauses in first-order logic. The authors consider the task to be *dynamic propositionalization* and induce the clauses iteratively. The quality of new candidates is evaluated by the performance of the candidate pattern (that is clausal) sets in a classifier (Naïve Bayes and an SVM, respectively). In contrast, in our formulation of the iterative pattern set mining task, patterns are (explicitly or implicitly) evaluated in terms of the instances they cover, not by their usefulness. The main ideas of both approaches are the same, however. Specifically, the approaches by Landwehr *et al.* use a heuristic method (beam search) for local pattern mining, possibly a direction for future research.

As we have argued before, iterative pattern set mining is inherently greedy since in the pattern set step not all possible extensions to the pattern set are explored. As we have hinted already in Section 6.1.2, however, this greedy technique could be extended to, for instance, a beam-search at the pattern set level as well – at the cost of increased computational complexity.

Part IV

Round-up

Summary and Future Work

In the introduction to this thesis, we sketched the KDD process and the role data mining plays in it. The goal of of KDD is finding "valid, novel, potentially useful and ultimately useful" patterns. While finding valid and novel patterns is topic that has been given extensive attention during the last two decades, the question of understandability has been less well researched.

To solve the problem of finding understandable pattern sets, we propose an additional data mining task, supplementing that of local pattern mining, the task of *pattern set mining*. This thesis is the first to explore the problem of pattern set mining in a principled and systematic way. Pattern set mining is becoming more important due to the efficient pattern mining techniques that exist in data mining and the need to organize the large result sets of local pattern mining operations. Depending on the tasks addressed, different characteristics of pattern sets are desired but a common theme is that such sets should be as small as possible, while still useful and interpretable.

Nevertheless, pattern set mining *has* been addressed before. Especially in the machine learning literature, but also in data mining systems that have been applied to tasks such as concept learning/classification, a variety of pattern set mining approaches has been proposed. Accordingly, there exist systems for concept learning, subgroup discovery, association rule summarization, and clustering that all perform pattern set mining, some as a post-processing step to local pattern mining, some integrated with local pattern mining. In most cases, however, pattern set mining has been presented as part of the overall system, that is, coupled to a specific local pattern mining technique. This integration has made it hard to identify pattern set mining as a separate task and to actively develop fitting solutions to the problem. Our work aimed at alleviating this situation.

Arguably the biggest obstacle to the development of dedicated pattern set mining techniques was that pattern set mining did not exist as a well-defined task, making it difficult to analyze and improve existing techniques, to see the relationships between such techniques, and to develop new ones. That is why we contributed a framework for pattern set mining in Chapter 1, to overcome this issue. To the best of our knowledge, this is the first time that such a framework has been proposed. Formulating pattern set mining as a distinct data mining task has provided us with a framework to reason about existing approaches and suggest directions of future research.

Different types of pattern sets

One of the key points of the framework is the realization that pattern sets can be grouped into three general groups, cf. Section 1.3.1:

- *Unordered sets*, which can be employed very flexibly since apart from the fact that they are somehow related to each other, due to being in the same pattern set, no restrictions on their interpretation exist. An unordered pattern set can, for instance, be a set of mined features that is used to represent data, or a set of unordered predictive rules that are combined via a voting strategy. This flexibility is traded off against the fact that unordered pattern sets cannot be mined by all pattern set mining techniques, while ordered and tree sets can be derived from unordered sets, given the right (partial) order.

- *Ordered sets*, in which each pattern's membership in the pattern set, and therefore their interpretation, is explicitly or implicitly influenced by the patterns that occur earlier it in the order. To a certain degree can this order can be helpful in interpreting patterns since it provides a context. On the other hand, this means that only the first pattern can be considered on its own. A common example for ordered sets are decision lists in which the first predictive rules that matches an instance predicts its class. The order that is imposed on the set can arise during the local pattern mining or pattern set mining step or be imposed afterwards, which also means that unordered sets can be turned into ordered ones.

- *Tree sets*, in which each pattern is influenced by its direct successor and this pattern's successors but in which patterns can be included that have no clear relation to each other. A decision tree is an example for a tree set, as are clustering trees. Tree sets can be ordered sets but not vice versa.

Each of these pattern set types lends itself to different interpretations and therefore different tasks. Additionally, not all of the existing pattern set mining approaches allows the mining of all types of pattern sets. We proposed a novel general exhaustive post-processing algorithm for mining unordered sets, the type of pattern set that has been considered least in the literature so far. This technique extends existing exhaustive approaches, making it possible to enforce a wider selection of constraints. It can therefore be used as an alternative to current solutions.

Certain heuristic techniques are well suited to mining ordered sets by post-processing, as we discussed in our work. Many of these techniques use an order on the result set of a local pattern mining operation to make pattern set mining computationally feasible. These orders are usually fixed, however, and while they are typically well-motivated, different choices remain possible. We proposed and evaluated the effects of different orders in several cases. Interpreting the results allowed us to suggest possible alternatives to existing choices in terms of used orders. Additionally, the use of orders can be forgone completely if, for instance, upper-bound calculations support the greedy mining process.

We showed how to modify ordered pattern set mining in this way and that the results abstract from the effects of heuristic parameters.

Exhaustive *versus* heuristic mining

In the case of post-processing of a local pattern mining operation, which is the currently typical data mining method of pattern set mining, heuristic mining leads to pattern sets of high quality, trading this quality off against efficiency, of course. This trade-off is necessary since the problem of combinatorial explosion, already pronounced in local pattern mining, becomes even worse for pattern sets.

The question of whether to use exhaustive or heuristic search is not only of interest in post-processing but also has to be considered when iterative pattern set mining is performed. In iterative pattern set mining, local pattern mining is performed repeatedly and the results combined into a pattern set. This leads to two questions whose interplay has an effect on the efficiency and effectiveness of iterative pattern set mining:

1. Does exhaustive local pattern mining lead to a more computationally costly pattern set mining operation?

2. Does heuristic local pattern mining lead to pattern sets approaching an optimal solution for the task at hand?

Iterative mining techniques originate in the machine learning community and have been proposed together with heuristic local pattern mining steps, which means that information about the effect of using exhaustive local pattern mining is rare. We added to the knowledge about this issue by comparing heuristic and exhaustive local pattern mining in several settings.

Two main insights can be derived from our experimental results. The first, new and surprising finding that arose is that exhaustive techniques are *not* at a disadvantage computationally. Instead, exhaustive mining often had lower computational complexity, and, additionally, give guarantees about the quality of mined patterns. This is connected to our second finding, that sequential covering is effective in mining a pattern set that is well-suited to the task at hand. Even though this advantage is less pronounced for descriptive tasks, for instance *subgroup discovery*, this strongly implies the applicability of sequential mining for pattern set mining purposes.

Post-processing *versus* iterative mining

This insight brings us to the final experimental aspect of our thesis. As we showed in our experimental evaluations, iterative mining allows us to assemble pattern sets that are very useful for the tasks for which they have been mined, be they predictive or descriptive. Iterative mining leads to smaller pattern sets, aiding understandability, that still have good accuracy if used as a classifier, or give a good description of the data.

Especially when combined with the exhaustive approaches to local pattern mining we discussed in this work, iterative mining shows a lot of potential and promises to be a very useful approach towards mining understandable, useful pattern sets from data.

Main contributions

So in summarizing the main contributions of our work:

1. We proposed a framework supplying the vocabulary needed for discussing pattern set mining and used it to characterize existing pattern set mining approaches and propose new ones.

2. Based on this analysis, we proposed improvements to pattern set mining techniques. These improvements range from upgrades, as in proposing a general exhaustive pattern set mining algorithm, to modifications, for instance to heuristic parameters used or the type of local pattern mining operation involved.

3. By experimentally evaluating these improvements, we gained understanding about the issues facing such modifications and collected evidence pointing towards the applicability of exhaustive local pattern mining and, more general, the use of iterative techniques for pattern set mining.

A smaller contribution lies in that the insights gained by our application and modification of sequential and parallel covering to pattern set mining can be applied to the areas in which these paradigms originate. They can be employed to gain a deeper understanding of the relation between local pattern mining and model formation, and issues regarding the effectiveness of these learning techniques.

Future Work

The framework we introduced allowed us to analyze pattern set mining techniques and particularly to decompose numerous existing techniques into their local pattern mining and pattern set mining components. While we used the experimental part of our work to explore different aspects of pattern set algorithms, the issues we highlighted do not exhaust the topic by far.

The main research direction for pattern set mining techniques concerns an approach that we have only encountered in a few, successful systems. KRIMP, RIPPER, ORIGAMI, and the technique proposed in (Rückert and Kramer 2007) all perform neighborhood search to a certain degree. This is a way of controlling computational complexity that differs from both heuristic post-processing and iterative mining. Since the interactions among patterns are often more complex than among components of a single pattern, this might very well be a better approach to traversing the search space than the uni-directional enumeration used in most existing pattern set mining techniques. To develop effective local

search methods for pattern set mining, it will be necessary to draw inspiration from the field of constrained local search, a research area that so far has had little impact on data mining.

The techniques we have described in our work should not be abandoned, however. In the conclusions to our experimental evaluations, we identified directions of improvement of current approaches that could be pursued in the future. The main directions could be, in no particular order:

- The improvement of iterative mining techniques for pattern set mining. This includes in particular the adaptation of iterative mining for unsupervised mining tasks and an increase in the complexity of the patterns that can be mined in each iteration. Such an approach could, after each iteration, split the data into equivalence classes, for instance, and iterate mining on each of them.

- The development of techniques that are situated between exhaustive and pure hill-climbing for pattern set mining. Exhaustive mining adopts a strategy of enumerating *all* pattern sets satisfying certain constraints and selecting the best one according to a quality measure. Hill-climbing, on the other hand, extends only a single pattern set and has therefore few mechanisms of escaping local optima. Adapting known heuristic techniques such as beam search will allow to efficiently mine pattern sets while at the same time improving the chance for reaching global optima.

- The improvement of the algorithmic details of existing exhaustive and heuristic techniques. In the exhaustive case this would, for instance, take the form of more effective constraints and specialized data structures. In heuristic mining this could, for instance, lead to developing upper-bound ordered alternatives to order-restricted hill-climbing.

Additionally, there are other techniques from the machine learning field for mining meaningful sets of patterns, chief among them different ways to exploit the data, such as in the *Boosting* and *Bagging* frameworks. The standard data mining setting, which we adhered to throughout the entire thesis, is to use the entire data available for mining and to perform the mining operation just once. In work that we did not report in detail here (Zimmermann and Bringmann 2009), we did some preliminary explorations in this direction. The insights we derived, that random selection can outperform aggregated quality measures, and that *Bagging*-like resampling is less useful than having independent subsets for mining, were somewhat surprising. They also indicate that a straight-forward application of machine learning techniques that use only parts of the data to pattern set mining is *not* always possible and different methods have to be developed.

The work we reported in this thesis has been largely experimental and while it resulted in several new insights, a theoretical analysis of these issues is certainly of interest. That exhaustive techniques are viable alternatives for mining operations that have only been approached heuristically so far, for instance, also

means that benchmarks regarding the quality of solutions can be established. In machine learning, principled usage of data allows one to give guarantees about the quality of resulting models. In data mining, on the other hand, a theory of the quality of found solutions that could be similar to computational learning theory, is almost completely absent so far. Notable exceptions include (Toivonen 1996) and (Scheffer and Wrobel 2002), which explored the use of sampling from databases for local pattern mining and derived bounds on the quality of resulting patterns. To this end, they modified the usual formulation of the PAC (**p**robably **a**pproximately **c**orrect) framework to accommodate descriptive mining. A direct application of the framework can be found in (Kudo et al. 2004), where the theoretical foundations of LPBoost are used to mine predictive graph patterns. An integration of such results and the derivation of additional theoretical relationships is still missing however, and would improve the understanding of pattern and pattern set mining the development of effective techniques.

A main problem for the post-processing approaches we discussed is hat all patterns mined in the local pattern mining step refer to a single background distribution, without recourse to other patterns. In iterative mining this problem is addressed but the solution constrains iterative mining to greedy approaches. The non-greedy alternative would ask for the extension of a pattern set with *all possible patterns* in each iteration, quickly leading to an combinatorial explosion due to the *implicit* enforcement of pattern set constraints. In both cases, however, a re-formulation of pattern set constraints as constraints for *local* patterns could be the solution to this problem, an approach that would be aided by theoretical results of the kind described in the preceding paragraph. The final goal of research in this direction would be *direct* mining of pattern sets in a single pass, using neither post-processing nor iterative mining.

There is no clear-cut answer to *where* to eat, since there is "no free lunch" after all (Wolpert et al. 1995). But we can make informed decisions about our choices and we hope that this thesis has given a general idea of how to make them. We are sophisticated now and we should act like it – mining sets of patterns is indispensable when it comes to discovering knowledge from databases and the better we are at doing it, the more valuable the results will be.

Bibliography

Adams, D.: 1979, *The Hitchhiker's Guide to the Galaxy*, Pan Books.

Agrawal, R., Imielinski, T. and Swami, A. N.: 1993, Mining association rules between sets of items in large databases, *in* P. Buneman and S. Jajodia (eds), *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ACM Press, pp. 207–216.

Agrawal, R. and Srikant, R.: 1994, Fast algorithms for mining association rules in large databases, *Proceedings of the 20th International Conference on Very Large Databases*, Morgan Kaufmann, Santiago de Chile, Chile, pp. 487–499.

Blake, C. and Merz, C.: 1998, UCI repository of machine learning databases. **URL:** *http://www.ics.uci.edu/∼mlearn/MLRepository.html*

Blockeel, H., Raedt, L. D. and Ramon, J.: 1998, Top-down induction of clustering trees, *in* J. W. Shavlik (ed.), *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 55–63.

Borgelt, C.: 2004, Recursion pruning for the apriori algorithm., *in* R. J. B. Jr., B. Goethals and M. J. Zaki (eds), *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.

Boulicaut, J.-F. and Jeudy, B.: 2001, Mining free itemsets under constraints., *in* M. E. Adiba, C. Collet and B. C. Desai (eds), *International Database Engineering & Applications Symposium, IDEAS '01*, pp. 322–329.

Breiman, L.: 1996, Bagging predictors, *Machine Learning* **24**(2), 123–140.

Breiman, L.: 2001, Random forests, *Machine Learning* **45**(1), 5–32.

Bringmann, B. and Karwath, A.: 2004, Frequent SMILES, *Lernen, Wissensentdeckung und Adaptivität, Workshop GI Fachgruppe Maschinelles Lernen, LWA*.

Bringmann, B. and Zimmermann, A.: 2005, Tree^2 - Decision trees for tree structured data., *in* A. Jorge, L. Torgo, P. Brazdil, R. Camacho and J. Gama (eds), *9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Springer, pp. 46–58.

Bringmann, B. and Zimmermann, A.: 2007, The chosen few: On identifying valuable patterns, *in* Ramakrishnan and Zaiane (2007), pp. 63–72.

Bringmann, B. and Zimmermann, A.: 2009, One in a million: picking the right patterns, *Knowledge and Information Systems* **18**(1), 61–81.

Bringmann, B., Zimmermann, A., De Raedt, L. and Nijssen, S.: 2006, Don't be afraid of simpler patterns, *in* Fürnkranz et al. (2006), pp. 55–66.

Bucila, C., Gehrke, J., Kifer, D. and White, W.: 2002, DualMiner: A dual-pruning algorithm for itemsets with constraints, *Proceedings of The Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton,Alberta,Canada.

Calders, T. and Goethals, B.: 2002, Mining all non-derivable frequent itemsets., *in* T. Elomaa, H. Mannila and H. Toivonen (eds), *Principles of Data Mining and Knowledge Discovery, 6th European Conference*, Springer, pp. 74–85.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. and Freeman, D.: 1988, Autoclass: A Bayesian classification system, *in* J. E. Laird (ed.), *Proceedings of the Fifth International Conference on Machine Learning*, Morgan Kaufmann, Ann Arbor, Michigan, USA, pp. 54–64.

Cheng, H., Yan, X., Han, J. and Hsu, C.-W.: 2007, Discriminative frequent pattern analysis for effective classification, *Proceedings of the 23rd International Conference on Data Engineering*, IEEE, pp. 716–725.

Cheng, H., Yan, X., Han, J. and Yu, P. S.: 2008, Direct discriminative pattern mining for effective classification, *Proceedings of the 24th International Conference on Data Engineering*, IEEE, pp. 169–178.

Clark, P. and Niblett, T.: 1989, The CN2 induction algorithm, *Machine Learning* **3**, 261–283.

Cohen, W. W.: 1995, Fast effective rule induction, *in* A. Prieditis and S. J. Russell (eds), *Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kaufmann, Tahoe City, California,USA, pp. 115–123.

Daylight Chemical Information Systems,Inc.: 2004, http://www.daylight.com/.

De Raedt, L. and Kramer, S.: 2001, The level wise version space algorithm and its application to molecular fragment finding, *in* B. Nebel (ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 853–862.

De Raedt, L. and Zimmermann, A.: 2007, Constraint-based pattern set mining, *Proceedings of the Seventh SIAM International Conference on Data Mining*, SIAM.

Demiriz, A., Bennett, K. P. and Shawe-Taylor, J.: 2002, Linear programming boosting via column generation, *Machine Learning* **46**(1-3), 225–254.

Fan, W., Zhang, K., Cheng, H., Gao, J., Yan, X., Han, J., Yu, P. S. and Verscheure, O.: 2008, Direct mining of discriminative and essential frequent patterns via model-based search tree, *in* Y. Li, B. Liu and S. Sarawagi (eds), *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 230–238.

Fayyad, U. M. and Irani, K. B.: 1993, Multi-interval discretization of continuous-valued attributes for classification learning, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Chambéry,France, pp. 1022–1029.

Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P.: 1996, From data mining to knowledge discovery in databases, *AI Magazine* **17**(3), 37–54.

Fisher, D. H.: 1987, Knowledge acquisition via incremental conceptual clustering, *Machine Learning* **2**(2), 139–172.

Fisher, D. H.: 1996, Iterative optimization and simplification of hierarchical clusterings., *Journal of Artificial Intelligence Research (JAIR)* **4**, 147–178.

Frank, E. and Witten, I. H.: 1999, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann.

Freund, Y. and Schapire, R. E.: 1997, A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences* **55**(1), 119–139.

Freund, Y. and Shapire, R.: 1996, Experiments with a new boosting algorithm, *in* L. Saitta (ed.), *Proceedings of the 13th International Conference on Machine Learning*, Morgan Kaufmann, pp. 148–156.

Fürnkranz, J. and Flach, P. A.: 2005, ROC 'n' rule learning-towards a better understanding of covering algorithms., *Machine Learning* **58**(1), 39–77.

Fürnkranz, J., Scheffer, T. and Spiliopoulou, M. (eds): 2006, *Knowledge Discovery in Databases: PKDD 2006,10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, Springer.

Galiano, F. B., Cubero, J. C., Sánchez, D. and Serrano, J.-M.: 2004, ART: A hybrid classification model, *Machine Learning* **54**(1), 67–92.

Gamberger, D. and Lavrač, N.: 2000, Confirmation rule sets, *in* D. A. Zighed, H. Komorowski and J. M. Zytkow (eds), *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, Lyon, France, pp. 34–43.

Gamberger, D. and Lavrac, N.: 2002, Expert-guided subgroup discovery: Methodology and application, *Journal of Artificial Intelligence Research (JAIR)* **17**, 501–527.

Geamsakul, W., Matsuda, T., Yoshida, T., Motoda, H. and Washio, T.: 2003, Performance evaluation of decision tree graph-based induction., *in* G. Grieser, Y. Tanaka and A. Yamamoto (eds), *Discovery Science, 6th International Conference*, Springer, Sapporo, Japan, pp. 128–140.

Gluck, M. A. and Corter, J. E.: 1985, Information, uncertainty, and the utility of categories, *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associate, Irvine, California, USA, pp. 283–287.

Guyon, I. and Elisseeff, A.: 2003, An introduction to variable and feature selection, *Journal of Machine Learning Research* **3**, 1157–1182.

Han, J., Cheng, H., Xin, D. and Yan, X.: 2007, Frequent pattern mining: current status and future directions, *Data Min. Knowl. Discov.* **15**(1), 55–86.

Han, J., Pei, J. and Yin, Y.: 2000, Mining frequent patterns without candidate generation, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ACM, Dallas,Texas,USA, pp. 1–12.

Hasan, M. A., Chaoji, V., Salem, S., Besson, J. and Zaki, M. J.: 2007, Origami: Mining representative orthogonal graph patterns, *in* Ramakrishnan and Zaiane (2007), pp. 153–162.

Höppner, F.: 2004, Local pattern detection and clustering, *in* K. Morik, J.-F. Boulicaut and A. Siebes (eds), *Proceedings of the Dagstuhl Workshop on Detecting Local Patterns*, Springer, Dagstuhl Castle, Germany, pp. 53–70.

Karwath, A. and De Raedt, L.: 2004, Predictive graph mining, *in* E. Suzuki and S. Arikawa (eds), *Proceedings of the 7th International Conference on Discovery Science*, Springer, Padova, Italy, pp. 1–15.

Kifer, D., Gehrke, J., Bucila, C. and White, W. M.: 2003, How to quickly find a witness., *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM, pp. 272–283.

Kilpeläinen, P.: 1992, *Tree Matching Problems with Applications to Structured Text Databases*, PhD thesis, University of Helsinki.

King, R. D., Sternberg, M. J. E. and Srinivasan, A.: 1995, Relating chemical activity to structure: An examination of ILP successes., *New Generation Comput.* **13**(3&4), 411–433.

Knobbe, A., Crémilleux, B., Fürnkranz, J. and Scholz, M.: 2008, From local patterns to global models: The lego approach to data mining, *in* J. Fürnkranz and A. Knobbe (eds), *From Local Patterns to Global Models: Proceedings of the ECML/PKDD-08 Workshop*, pp. 1–16.

Knobbe, A. J. and Ho, E. K. Y.: 2006a, Maximally informative k-itemsets and their efficient discovery., *in* T. Eliassi-Rad, L. H. Ungar, M. Craven and D. Gunopulos (eds), *KProceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data MiningDD*, ACM, pp. 237–244.

Knobbe, A. J. and Ho, E. K. Y.: 2006b, Pattern teams, *in* Fürnkranz et al. (2006), pp. 577–584.

Kohavi, R. and Kunz, C.: 1997, Option decision trees with majority votes, *in* D. H. Fisher (ed.), *Proceedings of the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 161–169.

Kudo, T., Maeda, E. and Matsumoto, Y.: 2004, An application of boosting to graph classification, *Advances in Neural Information Processing Systems 17*.

Landwehr, N., Kersting, K. and De Raedt, L.: 2005, nFOIL: Integrating Naïve Bayes and FOIL, *in* M. M. Veloso and S. Kambhampati (eds), *The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / The MIT Press, pp. 795–800.

Landwehr, N., Passerini, A., De Raedt, L. and Frasconi, P.: 2006, kFOIL: Learning simple relational kernels, *Proceedings of The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, AAAI Press.

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

Li, W., Han, J. and Pei, J.: 2001, CMAR: Accurate and efficient classification based on multiple class-association rules, *in* N. Cercone, T. Y. Lin and X. Wu (eds), *Proceedings of the 2001 IEEE International Conference on Data Mining*, IEEE Computer Society, San José, California, USA, pp. 369–376.

Liu, B., Hsu, W. and Ma, Y.: 1998, Integrating classification and association rule mining, *in* R. Agrawal, P. E. Stolorz and G. Piatetsky-Shapiro (eds), *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, New York City, New York, USA, pp. 80–86.

Liu, H. and Motoda, H.: 1998, *Feature Selection for Knowledge Discovery and Data Mining*, Kluwer Academic Publishers, Norwell,MA,USA.

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

McGarry, K.: 2005, A survey of interestingness measures for knowledge discovery, *The Knowledge Engineering Review* **20**(1), 39–61.

Michalski, R. S. and Stepp, R. E.: 1983, Learning from observation: Conceptual clustering, *Machine Learning, An Artificial Intelligence Approach* **1**, 331–363.

Mitchell, T.: 1997, *Machine Learning*, McGraw-Hill.

Murthy, S. K.: 1997, *On Growing Better Decision Trees from Data*, PhD thesis, John Hopkins University, Baltimore,Maryland,USA.

Nevins, A. J.: 1995, A branch and bound incremental conceptual clusterer., *Machine Learning* **18**(1), 5–22.

Ng, R. T., Lakshmanan, L. V. S., Han, J. and Pang, A.: 1998, Exploratory mining and pruning optimizations of constrained associations rules, *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pp. 13–24.

Perkowitz, M. and Etzioni, O.: 1999, Adaptive web sites: Conceptual cluster mining., *in* T. Dean (ed.), *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Stockholm, Sweden, pp. 264–269.

Quinlan, J. R.: 1990, Learning logical definitions from relations., *Machine Learning* **5**, 239–266.

Quinlan, J. R.: 1993, *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Ramakrishnan, N. and Zaiane, O. (eds): 2007, *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, IEEE Computer Society.

Rückert, U. and Kramer, S.: 2007, Optimizing feature sets for structured data, *in* J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic and A. Skowron (eds), *18th European Conference on Machine Learning*, Vol. 4701 of *Lecture Notes in Computer Science*, Springer, pp. 716–723.

Russell, S. J. and Norvig: 2003, *Artificial Intelligence: A Modern Approach (Second Edition)*, Prentice Hall.

Scheffer, T. and Wrobel, S.: 2002, Finding the most interesting patterns in a database quickly by using sequential sampling, *Journal of Machine Learning Research* **3**, 833–862.

Sese, J. and Morishita, S.: 2004, Itemset classified clustering, *in* J.-F. Boulicaut, F. Esposito, F. Giannotti and D. Pedreschi (eds), *Proceedings of the 8th European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, Pisa,Italy, pp. 398–409.

Shima, Y., Hirata, K. and Harao, M.: 2005, Extraction of frequent few-overlapped monotone dnf formulas with depth-first pruning., *in* T. B. Ho, D. Cheung and H. Liu (eds), *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference*, Vol. 3518 of *Lecture Notes in Computer Science*, Springer, pp. 50–60.

Siebes, A., Vreeken, J. and van Leeuwen, M.: 2006, Item sets that compress., *in* J. Ghosh, D. Lambert, D. B. Skillicorn and J. Srivastava (eds), *Proceedings of the Sixth SIAM International Conference on Data Mining*, SIAM.

Silberschatz, A. and Tuzhilin, A.: 1996, What makes patterns interesting in knowledge discovery systems, *IEEE Transactions on Knowledge and Data Engineering* **8**(6), 970–974.

Srinivasan, A., Muggleton, S., King, R. and Sternberg, M.: 1994, Mutagenesis: ILP experiments in a non-determinate biological domain, *in* S. Wrobel (ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming*, Vol. 237, Gesellschaft für Mathematik und Datenverarbeitung MBH, pp. 217–232.

Taouil, R., Pasquier, N., Bastide, Y. and Lakhal, L.: 2000, Mining bases for association rules using closed sets., *Proceedings of the 16th International Conference on Data Engineering*, IEEE Computer Society, p. 307.

The OpenBabel Software Community: 2003, Open Babel, http://openbabel.sourceforge.net/.

Toivonen, H.: 1996, Sampling large databases for association rules, *in* T. M. Vijayaraman, A. P. Buchmann, C. Mohan and N. L. Sarda (eds), *Proceedings of 22th International Conference on Very Large Data Bases*, Morgan Kaufmann, pp. 134–145.

van Leeuwen, M., Vreeken, J. and Siebes, A.: 2006, Compression picks item sets that matter, *in* Fürnkranz et al. (2006), pp. 585–592.

Vens, C., Struyf, J., Schietgat, L., Dzeroski, S. and Blockeel, H.: 2008, Decision trees for hierarchical multi-label classification, *Machine Learning* **73**(2), 185–214.

Weininger, D.: 1988, SMILES,a chemical language and information system 1. Introduction and encoding rules, *Journal of Chemical Information and Computer Science* **28**, 31–36.

Wolpert, D. H., Macready, W. G., H, D. and G, W.: 1995, No free lunch theorems for search.

Wrobel, S.: 1997, An algorithm for multi-relational discovery of subgroups, *in* J. Komorowski and J. Zytkow (eds), *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD '97)*, Springer-Verlag, Trondheim, Norway, pp. 78 – 87.

Wu, T., Chen, Y. and Han, J.: 2007, Association mining in large databases: A re-examination of its measures, *in* J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic and A. Skowron (eds), *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Vol. 4702 of *Lecture Notes in Computer Science*, Springer, pp. 621–628.

Yin, X. and Han, J.: 2003, Cpar: Classification based on predictive association rules, *in* D. Barbará and C. Kamath (eds), *Proceedings of the Third SIAM International Conference on Data Mining*, SIAM.

Zaki, M. J. and Aggarwal, C. C.: 2003, XRules: an effective structural classifier for XML data., *in* L. Getoor, T. E. Senator, P. Domingos and C. Faloutsos (eds), *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, Washington, DC, USA, pp. 316–325.

Zimmermann, A.: 2008, Ensemble-trees: Leveraging ensemble power inside decision trees, *in* J.-F. Boulicaut, M. R. Berthold and T. Horváth (eds), *Discovery Science, 11th International Conference*, Vol. 5255 of *Lecture Notes in Computer Science*, Springer, pp. 76–87.

Zimmermann, A. and Bringmann, B.: 2005, Ctc - correlating tree patterns for classification, *in* J. Han, B. W. Wah, V. Raghavan, X. Wu and R. Rastogi (eds), *Proceedings of the Fifth IEEE International Conference on Data Mining*, IEEE, Houston, Texas, USA, pp. 833–836.

Zimmermann, A. and Bringmann, B.: 2009, Aggregated subset mining, *in* T. Theeramunkong, B. Kijsirikul, N. Cercone and T. B. Ho (eds), *Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference*, Vol. 5476 of *Lecture Notes in Computer Science*, Springer, pp. 664–672.

Zimmermann, A. and De Raedt, L.: 2009, Cluster-Grouping: From subgoup discovery to clustering, *Machine Learning Journal* . Accepted for publication.

Publication List

Journal Articles

- Albrecht Zimmermann and Luc De Raedt. *Cluster-Grouping: From Subgoup Discovery to Clustering*. Machine Learning Journal, 2009. *Accepted for publication*

- Björn Bringmann and Albrecht Zimmermann. *One in a Million: Picking the Right Patterns*. Knowledge and Information Systems 18 (1), pp. 61–81, 2009.

Conferences and Workshops, Published in Proceedings

- Albrecht Zimmermann and Björn Bringmann. *Aggregated Subset Mining (Short Paper)*. In Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2009), Bangkok, Thailand, 2009, volume 5476 of Lecture Notes in Computer Science, pp. 664–672, Springer.

- Albrecht Zimmermann. *Ensemble-Trees: Leveraging Ensemble Power Inside Decision Trees*. In Proceedings of the 11th International Conference on Discovery Science (DS 2008), Budapest, Hungary, 2008, volume 5255 of Lecture Notes in Computer Science, pp. 76–87, Springer.

- Björn Bringmann and Albrecht Zimmermann. *The Chosen Few: On Identifying Valuable Patterns*. In Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), Omaha, Nebraska, USA, 2007, pp. 63–72, IEEE Computer Society.

- Luc De Raedt and Albrecht Zimmermann. *Constraint-Based Pattern Set Mining*. In Proceedings of the Seventh SIAM International Conference on Data Mining (SDM 2007), Minneapolis, Minnesota, USA, 2007, SIAM.

- Björn Bringmann and Albrecht Zimmermann and Luc De Raedt and Siegfried Nijssen. *Don't Be Afraid of Simpler Patterns*. In Proceedings of

the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2006), Berlin, Germany, 2006, pp. 55–66, volume 4213 of Lecture Notes in Computer Science, Springer.

- Albrecht Zimmermann and Björn Bringmann. *CTC - Correlating Tree Patterns for Classification(Short Paper).* In Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), Houston, Texas, USA, 2005, pp. 833–836, IEEE Computer Society.

- Björn Bringmann and Albrecht Zimmermann. *Tree^2 - Decision Trees for Tree Structured Data.* In Lernen, Wissensentdeckung und Adaptivität (LWA) 2005, GI Workshops, Saarbrücken, Germany, 2005, pp. 139–144, DFKI.

- Björn Bringmann and Albrecht Zimmermann. *Tree^2 - Decision Trees for Tree Structured Data.* In Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005), Porto, Portugal, 2005, pp. 46–58, volume 3721 of Lecture Notes in Computer Science, Springer.

- Albrecht Zimmermann and Luc De Raedt. *CorClass: Correlated Association Rule Mining for Classification.* In Proceedings of the 7th International Conference on Discovery Science (DS 2004), Padova, Italy, 2004, pp. 60–72, volume 3245 of Lecture Notes in Computer Science, Springer.

- Albrecht Zimmermann and Luc De Raedt. *Cluster-Grouping: From Subgroup Discovery to Clustering (Extended Abstract).* In Proceedings of the 15th European Conference on Machine Learning (ECML 2004), Pisa, Italy, 2004, pp. 575–577, volume 3201 in Lecture Notes in Computer Science, Springer.

Book Chapters

- Albrecht Zimmermann and Luc De Raedt. *Inductive Querying for Discovering Subgroups and Clusters.* In Jean-François Boulicaut and Luc De Raedt and Heikki Mannila (Eds.): Constraint-Based Mining and Inductive Databases, volume 3848 of Lecture Notes in Computer Science, Springer.

Biography

Albrecht Zimmermann was born on March 24, 1976, in Leipzig, Germany. He attended the Ernst-Thälmann-Schule and the mathematically-natural-science specialized Wilhelm-Ostwald-Gymnasium in Leipzig. After moving to Bremen in 1993, he graduated with "Abitur" from the Schulzentrum Sekundarstufe II "Lange Reihe" in June 1995. Following compulsory civil service, he began studying computer science at the Philipps-Universität in Marburg, Germany, in October 1996. After obtaining the "Vordiplom" in August 1998, he spent two semesters at the University of Tennessee in Knoxville, TN, USA, and after returning, resumed his studies at the Albert-Ludwigs-Universität in Freiburg, Germany. He received his "Diplom" in computer science in July 2003, and joined the machine learning group in Freiburg to begin Ph.D. work in the area of pattern mining, supervised by Prof. Luc De Raedt. In January 2007, he relocated with the group of Prof. Luc De Raedt to the DTAI (Declarative Talen en Artificiëele Intelligetie) group at the Katholieke Universiteit Leuven, Belgium. In May 2009, he will defend his Ph.D. thesis on "Mining Sets of Patterns" at the Katholieke Universiteit Leuven.