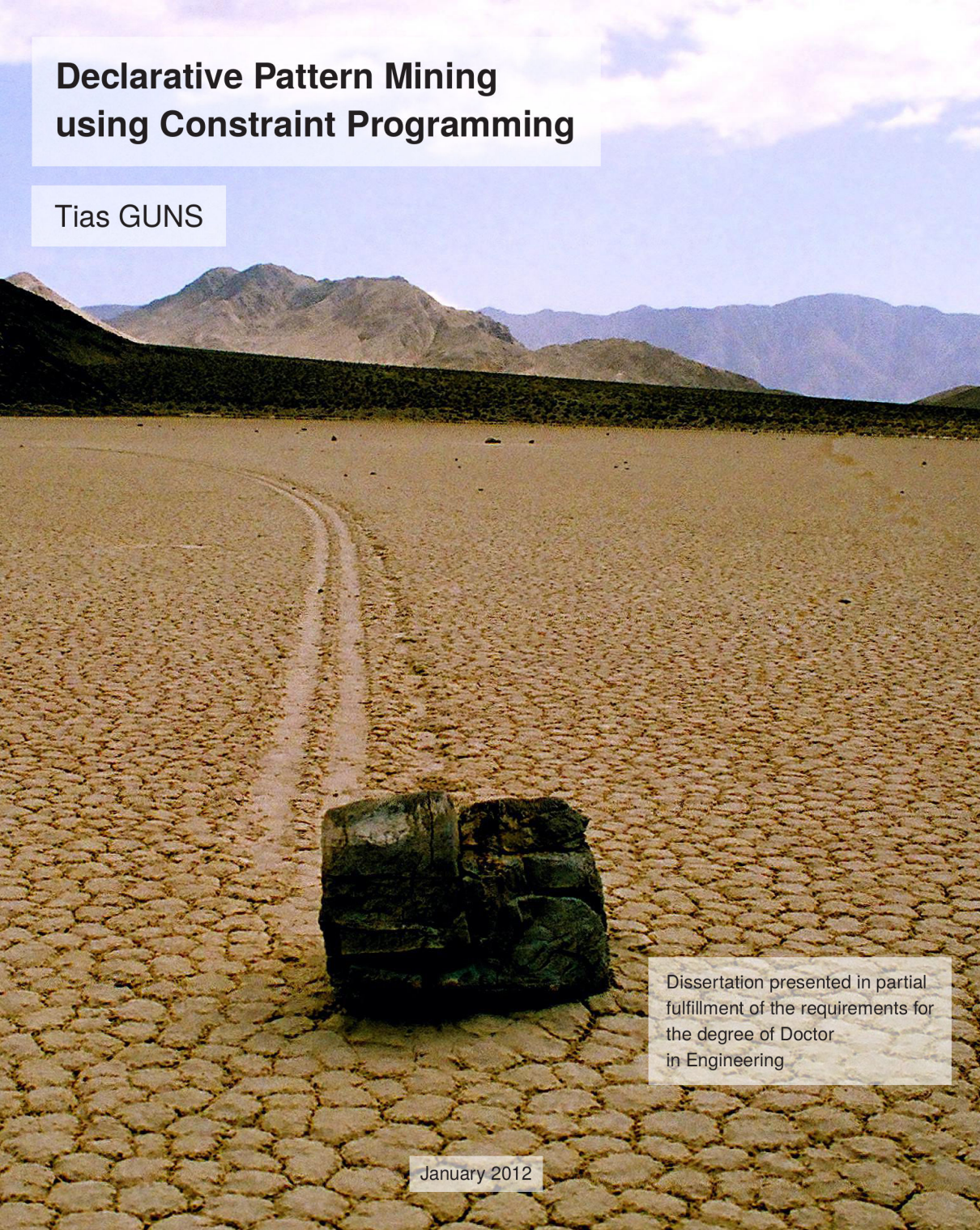
KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Declarative Pattern Mining using Constraint Programming

Tias GUNS



Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

January 2012

Declarative Pattern Mining using Constraint Programming

Tias GUNS

Jury:
Prof. dr. Carlo Vandecasteele, chair
Prof. dr. Luc De Raedt, promotor
Dr. Siegfried Nijssen, co-promotor
Prof. dr. Bettina Berendt
Prof. dr. ir. Maurice Bruynooghe
Prof. dr. Patrick De Causmaecker
Prof. dr. Barry O'Sullivan
(University College Cork, Ireland)
Prof. dr. Pascal Van Hentenryck
(Brown University, Providence, USA)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

January 2012

D/2012/7515/7
ISBN 978-94-6018-468-0

Abstract

The goal of pattern mining is to discover patterns in data. Many techniques have been proposed for this task, differing in the type of patterns they find. To ensure that only patterns of interest are found, a common approach is to impose *constraints* on the patterns. Constraint-based mining systems exist in which multiple constraints can be specified. However, combining constraints in new ways or adding complex constraints requires changing the underlying algorithms. A truly general approach to constraint-based pattern mining has been missing.

In this thesis we propose a general, *declarative* approach to pattern mining based on constraint programming. In a declarative approach one specifies *what* patterns need to be found, instead of algorithmically specifying *how* they must be found. Constraint programming offers a methodology in which a problem is stated in terms of constraints and a generic solver finds the solutions.
A first contribution of this thesis is that we show how constraint programming can be used to solve constraint-based, closed and discriminative itemset mining problems as well as combinations thereof. A second contribution is that we demonstrate how the difference in performance between general constraint solvers and specialised mining algorithms can be reduced. A third contribution is the introduction of the k-pattern set mining problem, which involves finding a set of k patterns that *together* satisfy constraints. We propose a high-level declarative language for k-pattern set mining as well as a transformation of this language to constraint programming. Finally we apply our declarative pattern mining framework on a challenging problem in bioinformatics, namely *cis*-regulatory module detection. For this application, the ability to add domain-specific constraints and to combine them with existing constraints is essential.

Hence we investigate, for the first time, how constraint programming can be used in pattern mining. We conclude on this promising approach with several remaining challenges.

Beknopte samenvatting

Tegenwoordig kunnen *pattern mining* algoritmen snel grote hoeveelheden patronen vinden in gegevens. Om te garanderen dat enkel interessante patronen gevonden worden, is het gebruikelijk om beperkingen of *constraints* aan de patronen op te leggen. Er bestaan constraint-gebaseerde mining algoritmen die verscheidene soorten constraints ondersteunen. Echter, het toevoegen van nieuwe complexe constraints of het combineren van constraints op onvoorziene manieren is in deze systemen enkel mogelijk door het onderliggende algoritme aan te passen. Een algemene aanpak voor constraint-gebaseerd pattern mining ontbreekt.

In deze thesis stellen we een algemene *declaratieve* aanpak tot pattern mining voor, door middel van *constraint programming*. In een declaratieve aanpak specifieert men *wat* er gevonden wordt, in plaats van algoritmisch te bepalen *hoe* patronen te vinden. In constraint programming gebeurt dit door middel van *constraints*, waarbij een automatische *solver* deze beperkingen gebruikt om een oplossing te vinden.
Een eerste bijdrage van deze thesis is dat we tonen hoe constraint programming gebruikt kan worden om verschillende pattern mining taken op te lossen. In een tweede bijdrage tonen we hoe het verschil in efficiëntie tussen een algemene solver en een specifiek algoritme kan worden weggewerkt. Een derde bijdrage is de introductie van het k-pattern set mining probleem, waarbij een verzameling van k patronen wordt gezocht die *samen* aan constraints voldoen. We stellen hiervoor een hoog-niveau declaratieve taal voor alsook een transformatie van deze taal naar constraint programming. Tot slot passen we ons declaratief pattern mining framework toe op een probleem uit de bio-informatica, namelijk het vinden van *cis*-regulerende modules. Voor deze toepassing is het toevoegen van domein-specifieke constraints en het combineren hiervan met bestaande constraints cruciaal.

Ons onderzoek verkent voor het eerst het gebruik van constraint programming in data mining. Dit is een beloftevolle aanpak die verdere vragen oproept.

Acknowledgements

There are many people that have helped me, directly or indirectly, with creating this thesis. To all of you, a big THANK YOU.

First and foremost I want to thank Luc De Raedt and Siegfried Nijssen, my promotor and co-promotor; without their vision and expertise this thesis would certainly not be what it is today. I feel very lucky that they have both been so interested and involved in the topic from the start, even though it meant that after every meeting, the amount of papers to read doubled while my self-esteem halved. I'm thankful to Elisa Fromont for her advice and guidance on these and other *soft* aspects of research.
I'm very grateful to be able to work with both Luc and Siegfried: Luc taught me to aim high, to think about the bigger picture, and basically, to be an Artificial Intelligence researcher more than a data mining researcher (which I only realised after visiting a major AI conference in 2010), while Siegfried taught me to think like a data miner, to care about nice formalisms and to have an eye for all the fine details. They are both passionate about good writing and drawing new links between existing works, and I increasingly find myself in the same situation.

I'm also grateful to Pascal Van Hentenryck and Barry O'Sullivan for accepting to be in my jury, they both had a big influence on this work long before the writing of this thesis. Pascal's ground-breaking research on combining constraint programming with local search has always been an inspiration for our work on combining constraint programming with pattern mining. I first met Barry at a Constraint Programming summer school in 2008, that summer school and our discussions really motivated me to not just be a user of CP, but to become a bit of a CP researcher myself. I'm also thankful to Bettina Berendt, Patrick De Causmaecker and Maurice Bruynooghe for being part of my jury and for all the critical but constructive feedback; to Carlo Vandecasteele for chairing the defence and of course to Luc and Siegfried for all their extensive comments on this thesis as it evolved. Thanks also to Matthijs and Siegfried for

their advice on how to make the cover more aesthetically pleasing. I also want to thank Jesse Davis for challenging me to finish a first version of this thesis before my birthday. I did not believe it would be possible, but it transformed the writing process from a daunting task into something manageable.

I've had the pleasure to share an office with many nice colleagues such as Albrecht, Anton, Beau, Bjoern, Joris, Leander, Matthijs en Thanh. My first office mates were Leander and Beau. Leander basically introduced me to the *life of a PhD student*, including "going for coffee" and staying up-to-date with what is going on in the group, playing ping pong to fend of frustration, going to talks and receptions, and organizing fun group activities. I have also greatly enjoyed our weekly running trips over lunchtime, together with Eduardo.
During the other lunch breaks, I've enjoyed many fine Alma lunches with Angelika, Anton, Daan, Guy, Hendrik, Ingo, Jan S., Kurt DG., Leander, Matthijs, McElory, Siegfried, Thanh and pretty much every other colleague. "Going to the Alma" really made the days a lot more social, diverse and interesting, so thanks for that.

Although officially my co-promotor, I always considered Siegfried more to be a colleague and friend. We've been sitting next to each other in the office for about 3 years now, talking about work, life, open source software and whatnot, sharing the load on FvI exercise sessions, pulling all-nighters to make deadlines and travelling to conferences and their surroundings together. I have always experienced our collaborations as pleasant yet effective, which creates a wonderful working environment.

Apart from colleagues, I'd like to especially thank my friends and fellow computer scientists Johan and Merel, who are actually interested when I talk nerdy or in details about work. Also a big thanks to my other friends, even just for not giving up on me when I slip under the radar for a while. More thanks are due to my circus friends, for balancing in on the artistic side of life. Especially to Kris and Lien for being delightfully unconventional and refreshing and to Mieke for the acrobatic trainings and performances we endeavoured on.

And then there is Nora, my love. A simple acknowledgement could never do justice to your contributions. You keep me safe from overdoing myself and from spending all of my free time on work, FOSDEM, the circus school and other projects I tend to get sucked in. You inspire me to live a diverse, happy and balanced life. Thank you for everything.

Finally, I wish to thank my parents for creating an environment in which I can freely pursue my dreams. Perhaps most remarkable is how rarely I have been conscience about this support or how it helped me grow.

Tias, January 2012

Contents

List of Figures

List of Tables

List of Symbols

$[\cdot]$	1 if the constraint within brackets is satisfied, 0 if not
$\lessgtr$	Shorthand for any of $\{\leq, <, =, \neq, >, \geq\}$
$dist(\pi^a, \pi^b)$	Distance function on two patterns
$\varphi_{\mathcal{D}}(I)$	The cover of itemset I in dataset $\mathcal{D}$
Π	A pattern set
π	A pattern, in case of itemset mining $\pi = (I, T)$
$\mathcal{S}$	The set of all transaction identifiers
$\mathcal{S}^+$	The set of all *positive* transaction identifiers
$\mathcal{S}^-$	The set of all *negative* transaction identifiers
D	Domain of variables in a constraint satisfaction problem
$D(v)$	Domain of variable v in a constraint satisfaction problem
I	A set of items
I_i	Boolean variable representing whether $i \in I$
$p(\pi, \mathcal{D})$	A constraint p on a pattern for given data
T	A set of transactions
T_t	Boolean variable representing whether $t \in T$
$\mathcal{D}$	Data
$\mathcal{D}_{ti}$	Boolean representing whether $(t, i) \in \mathcal{D}$
$\mathcal{I}$	The set of all items
$\mathcal{L}$	Pattern Language

Chapter 1

Introduction

This thesis links the research fields of data mining and constraint programming. We introduce data mining and constraint programming in turn and provide an overview of the contributions of the thesis and its general structure.

1.1 Data Mining

Data mining is concerned with discovering knowledge in data. With the advent of the internet and digital information, the amount of data that is generated and stored has taken on enormous proportions. Today, the key problem is not how to store the information, but how to analyse it. This has been nicely described by futurist John Naisbitt as 'we are drowning in data but starving for knowledge'. Data mining techniques aim at leveraging the available data and extracting knowledge from it. The discovered knowledge can take many forms. In *pattern mining* one is specifically interested in patterns, which are substructures that appear in the data that is analysed. For example, patterns can be sentences in a collection of texts, fragments in a set of molecules or search terms in a history of online search queries.

In itemset mining, a pattern is a set of items, also called an *itemset*. This term comes from the market basket analysis domain that inspired initial research on pattern mining. In market basket analysis, the data consist of a large number of customer purchases, called transactions. Each transaction consists of a number of *items* that a customer purchased together. For example, one transaction records that a customer purchased beer and chips and bread, the

next transaction records the purchase of chips, cheese and bread, etc. The goal then is to find interesting sets of items in these transactions, for example that chips and bread are frequently bought together. The discovered patterns can be used for marketing purposes, for configuring store layouts, for studying customer behaviour, and so forth.

Since its inception, itemset mining techniques have been used in a wide range of applications including document analysis [Holt and Chung 1999], anomaly detection [Lee et al. 2000], web usage mining [Tan and Kumar 2001] and numerous bioinformatics problems such as micro-array analysis [Becquet et al. 2002; Fujibuchi et al. 2009; Alves et al. 2010] and motif discovery [Huang et al. 2007; Sandve et al. 2008; Sun et al. 2009]. Itemset mining techniques can be used on any type of data in which one can identify transactions and items. In that case, out-of-the-box mining systems can find patterns that can lead to novel insights.

In pattern mining, a great deal of research effort has been put into developing fast and scalable algorithms. There have even been competitions for solving the basic frequent itemset mining problem as fast as possible [Goethals and Zaki 2004]. Nowadays, it is possible to mine large datasets in a matter of minutes and to discover a huge amount of patterns in the process. However, such a large collection of patterns is hard to interpret and needs further processing in order to extract relevant knowledge from it. Hence, it is generally recognized that the major bottleneck no longer lies in finding all frequent patterns efficiently, but rather in finding a *small set of interesting patterns* [Han et al. 2007].

Despite the existence of numerous approaches, it is still unclear how to find a small set of relevant and interesting patterns. Part of the difficulty is that 'interestingness' can be defined in many different ways. A popular approach is to express interestingness through constraints on data and patterns. This led to the development of new techniques such as constraint-based mining, condensed representations and pattern set mining. *Constraint-based mining* [Mannila and Toivonen 1997; Ng et al. 1998; Bayardo et al. 2000; Soulet and Crémilleux 2005; Bonchi and Lucchese 2007] reduces the large number of patterns by imposing additional constraints on the item- and transaction-sets. A challenge here is how to use these constraints effectively during the mining. *Condensed representations* [Mannila 1997; Bayardo 1998; Pasquier et al. 1999; Calders and Goethals 2007] aim at reducing redundancy among patterns without sacrificing too much information. In *pattern set mining* [De Raedt and Zimmermann 2007; Bringmann et al. 2010], the goal is to mine for a small set of patterns that *together* satisfy the given constraints. Different constraints can be imposed, both on individual patterns (locally) and on the entire pattern set (globally). Example pattern set mining tasks include clustering all patterns, summarizing the data or discriminating positive from negative examples in labeled data.

Unfortunately, the above mentioned approaches are task-specific and have largely an algorithmic focus. Algorithms have been designed to support a predefined number of constraints and have been optimized for high performance and scalability. Extending the algorithms to support novel constraints or combinations of constraints is non-trivial and requires a deep understanding of the underlying data structures and optimisations. For example, adapting constraint-based mining systems to incorporate condensed representations or to solve a pattern set mining task would require fundamental changes to the underlying algorithms. Hence, a truly general constraint-based mining approach is missing.

Creating a general framework for data mining is considered as one of the biggest challenges in the field [Mannila 2000; Yang and Wu 2006]. One of the most promising approaches is by Imielinski and Mannila [1996] who propose the concept of *inductive databases* as a general framework for data mining. An inductive database is a database that contains the data as well as (conceptually) the patterns that can be found in the data. In the same manner as data is retrieved from a database by querying, so can patterns be queried from the inductive database. This concept of data mining by means of querying is a powerful paradigm as it separates the specification of the constraints (the query) from the actual mining (the search). A number of data mining query languages [Meo et al. 1998; Imielinski and Virmani 1999; Tang and MacLennan 2005] and inductive database systems [Blockeel et al. 2008; Bonchi et al. 2009] exist, but they are restricted to the few primitives embedded in that specific language (see [Blockeel et al. 2011] for a practical comparison).

The interest in a general framework for expressing arbitrary constraints and effectively using them during search is shared by a completely different research community, namely, the constraint programming community. We discuss constraint programming in the following section.

1.2 Constraint Programming

Constraint programming is a general methodology for solving combinatorial problems, and involves finding a combination of elements adhering to a set of constraints. The essence of constraint programming can be described by the slogan

$$\text{Constraint Programming} = \text{Model} + \text{Search}$$

The model is a specification of a problem in terms of constraints on variables. For example, the constraint $X + Y > Z$ on variables X, Y and Z states that the

sum of X and Y must be larger than the value of Z. A solution is an assignment to the variables that satisfies all constraints; a solution to the constraint above would be $X = 4, Y = 6, Z = 8$. Search is performed by a solver that infers from the constraints, during search, what parts of the search space can be skipped.

The constraint programming methodology is by nature declarative: the user declares *what* constraints need to be satisfied. It is the burden of the solver to determine *how* the solutions are found. Constraint programming has successfully been used in a wide range of applications ranging from scheduling, planning and vehicle routing to bioinformatics and more [Rossi et al. 2006; Benhamou et al. 2007].

The constraint specification, or *model*, is typically specified in a constraint modeling language. Many such languages exist, for example OPL [Van Hentenryck 1999], MiniZinc [Nethercote et al. 2007] and Essence [Frisch et al. 2008]. Each modeling language supports a number of *variable* types, such as Boolean, integer or set variables. On each type of variable, a number of predefined general *constraints* can be imposed. Typical constraints include arithmetic constraints, extensional constraints, sequence constraints and combinatorial constraints. Additionally, new constraints can often be modeled by decomposing them into other constraints. The key concept of a modeling language is that it is high-level and declarative.

A *constraint solver* will use the constraints to reduce the search space. The size of the search space is determined by the *domain* of the variables, where the domain of a variable represents the values that it can still take. The key principle for reducing the search space is *propagation*. Propagation is the act of reducing the domain of a variable based on the constraints and the domains of other variables. Every constraint in the modeling language has a corresponding propagator in the solver. The *propagator* of a constraint not only enforces that a constraint is satisfied but it also pro-actively removes values that would violate it. The beauty of this approach is that every propagator is independent; it neither depends on other propagators nor does it enforce an order in which propagators need to be called. A propagator simply takes domains as input and produces the same or smaller domains as output. This allows constraints and propagators to be reused across different problems and applications.

The generality of constraint programming stems from its principled separation of concerns regarding modeling and solving. A model can be changed slightly or extensively, without the need to change the underlying solver. In case of solver independent modeling languages, one model can be solved by multiple solvers as well. Additionally, extending the solver with a new constraint only requires the addition of a single new propagator. The constraint can subsequently be used together with all other existing constraints. This flexibility makes constraint

programming an incredibly powerful platform for all kinds of purposes.

1.3 Contributions

Given the extensive use of constraints in data mining, the *question arises whether constraint programming can be used to solve data mining problems.* This thesis offers a positive answer to this question. The **main contribution** of this thesis is a declarative approach to itemset mining using methods from constraint programming.

In this thesis, we focus on the following three research questions:

Q1 What can constraint programming offer to data mining?

Q2 To what range of data mining problems can constraint programming be applied?

Q3 What are the limitations of using constraint programming for data mining?

The main insights and contributions of this thesis with respect to **Q1**, *what can constraint programming offer to data mining*, are the following:

- We use the declarative modeling principle of constraint programming to define a general language in which many itemset mining problems can be expressed. The key insight in this respect is that constraint programming can offer a **unifying language** for many data mining problems.
- A declarative language offers a great deal of flexibility for modeling problems and constraints. As a result, a contribution of this thesis is the constraint-programming based CP4IM framework, which is the **most flexible itemset mining system** to date. As we will show, our framework supports a wider range of constraints than other state-of-the-art mining systems, and additionally permits those constraints to be freely combined.
- Another contribution is that we show how the propagation principles of constraint programming can be used to **improve the pruning** of certain pattern mining specific constraints. Hence our work demonstrates that the concept of domains and propagation can shed new light on existing data mining problems and even improve the state-of-the-art.

With regard to **Q2**, *to what range of data mining problems can constraint programming be applied*, the main findings are the following:

- A contribution is that we **specify many mining problems** in a general constraint modeling language, including many types of constraint-based itemset mining, condensed representations, discriminative itemset mining and pattern set mining. The latter is general enough that it can be instantiated to many known mining and learning problems, such as concept learning, conceptual clustering and tiling.
- We also present a **real-life application** of the CP4IM framework on a challenging problem in bioinformatics. This requires the addition of domain-specific constraints to the framework. It demonstrates the applicability of the approach beyond well-studied data mining problems.
- A general insight of our work is that if a certain data mining task can be modeled in constraint programming, **many variants** of that task **can be modeled as well**. They usually only differ in the constraints that need to be enforced, hence changing or adding a few constraints is often sufficient.

This then leads to question **Q3**, *what are the limitations of using constraint programming for data mining*:

- On the one hand we show that out-of-the-box constraint solvers do **not perform well** compared to specialised mining systems, especially when dealing with large datasets or weakly constrained settings. On the other hand, in very constrained settings we show that out-of-the-box constraint solvers can outperform such specialised systems.
- A contribution in this respect is that we demonstrate how to construct an integrated constraint solver, with specialised data and variable representations common in itemset mining. This **integrated solver largely closes the performance gap** between general constraint solvers and specialised mining algorithms.
- A further insight throughout this thesis is that many mining problems can be modeled in constraint programming, but that effective search is **only possible when effective propagators** can be defined for the constraints.

A final contribution is that all of the data used and software developed in this thesis are available under an **open-source** license at:
http://dtai.cs.kuleuven.be/CP4IM/.

1.4 Structure of the thesis

Chapter 2 presents background information on itemset mining and constraint programming.

Chapter 3 introduces the core framework. It draws the link between constraint-based itemset mining and constraint programming, and shows how to use principles from constraint programming for constraint-based mining. We start with the most basic setting, the frequent itemset mining problem. Subsequently, for many popular variants such as closed itemset mining, maximal itemset mining, cost-based itemset mining and discriminative itemset mining we demonstrate how to formulate them in the same framework as well. By showing how to express the individual mining constraints, other existing as well as new constraint-based mining problems can be modeled and solved. In this way, the principles of constraint programming extend and improve the state-of-the-art in itemset mining, and itemset mining can provide a new application area for constraint programming. The chapter consists of the research previously published in the following papers:

L. De Raedt, T. Guns, and S. Nijssen. *Constraint programming for itemset mining*, in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-08), Las Vegas, Nevada, USA, 2008.

S. Nijssen, T. Guns, and L. De Raedt. *Correlated itemset mining in ROC space: A constraint programming approach*, in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-09), Paris, France, 2009.

L. De Raedt, T. Guns, and S. Nijssen. *Constraint programming for data mining and machine learning*, in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), Atlanta, Georgia, USA, 2010.

T. Guns, S. Nijssen, and L. De Raedt. *Itemset mining: A constraint programming perspective*, Artificial Intelligence, volume 175(12-13), 2011.

Chapter 4 studies the difference in performance between specialised mining systems and generic constraint solvers that was observed in the previous chapter. We compare the specialised techniques and data structures used in mining algorithms with those of the constraint solver used previously. This leads us to create an integrated constraint solver that achieves runtime and scalability performances similar to specialised data mining algorithms, while

maintaining the general principles of constraint programming. The chapter was previously published as:

S. Nijssen and T. Guns. *Integrating constraint programming and itemset mining*, in Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD-10), Barcelona, Spain, 2010.

In the next chapters, we extend the constraint programming for itemset mining framework in multiple directions.

Chapter 5 investigates the pattern set mining problem from a constraint programming point of view. The goal in patterns set mining is not to find individual patterns, but to find a small set of coherent patterns that satisfy given constraints. Many known data mining and machine learning tasks can be seen as instances of pattern set mining, for example the clustering or summarisation of data, or the discovery of redescriptions. We show for each of these tasks how to formulate them in our framework. Additionally, we investigate a novel one-step approach to solve these tasks, which teaches us valuable lessons about the relation between local and global constraints. The chapter is based on the following journal paper:

T. Guns, S. Nijssen, and L. De Raedt. *k-pattern set mining under constraints*, IEEE Transactions on Knowledge and Data Engineering, To appear, 2011. Also as Technical Report CW596, Oct 2010.
Winner of the Innovation Award at the Lorentz workshop on Mining Patterns and Subgroups, 2010.

Chapter 6 takes the pattern set mining task of concept learning and compares many proposed pattern set search strategies, including the novel one-step approach of the previous chapter. All strategies are implemented using our general constraint programming based framework, allowing for a fair and in-depth comparison. The chapter is based on the following paper:

T. Guns, S. Nijssen, and L. De Raedt. *Evaluating pattern set mining strategies in a constraint programming framework*, in Advances in Knowledge Discovery and Data Mining, 15th Pacific-Asia Conference (PAKDD-11), Shenzen, China, 2011.

Chapter 7 presents a bioinformatics application of our framework, involving the detection of *cis*-regulatory modules in genomic sequences. Itemset mining techniques have been applied to this problem before, though this required preprocessing the data in ways that have negative effects on the quality of the

result. Instead, we use our framework to incorporate domain-specific knowledge and constraints in the mining process, without having to significantly change the other constraints. This was published in the following paper:

> T. Guns, H. Sun, K. Marchal, and S. Nijssen. *Cis-regulatory module detection using constraint programming*, in Proceedings of the IEEE International Conference on Bioinformatics & Biomedicine (BIBM-10), Hong Kong, China, 2010.

In a concluding chapter, the thesis is summarized and opportunities for future work are discussed.

Chapter 2

Background

This chapter[1] provides the necessary background on pattern mining and constraint programming. The concepts and techniques explained will be used throughout the rest of the thesis.

2.1 Pattern Mining

In general, pattern mining is concerned with finding all patterns π that adhere to some constraint p [Mannila and Toivonen 1997]. The possible patterns are defined by a pattern language $\mathcal{L}$. The constraint p is typically a conjunction of multiple constraints that can be defined on data $\mathcal{D}$. The mining problem is then concerned with finding the set of all patterns π in the language $\mathcal{L}$ that satisfy the constraint, that is, the theory:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\pi \in \mathcal{L} \mid p(\pi, \mathcal{D}) \text{ is true}\}. \tag{2.1}$$

Numerous approaches to pattern mining have been developed to effectively find the patterns adhering to a set of constraints; a well-known example is the problem of finding frequent itemsets.

[1] Based on the background material in journal papers [Guns et al. 2011b] and [Guns et al. 2011c].

$\mathcal{D}$	Itemset
T_1	{B}
T_2	{E}
T_3	{A,C}
T_4	{A,E}
T_5	{B,C}
T_6	{D,E}
T_7	{C,D,E}
T_8	{A,B,C}
T_9	{A,B,E}
T_{10}	{A,B,C,E}

Tid	A	B	C	D	E
1	0	1	0	0	0
2	0	0	0	0	1
3	1	0	1	0	0
4	1	0	0	0	1
5	0	1	1	0	0
6	0	0	0	1	1
7	0	0	1	1	1
8	1	1	1	0	0
9	1	1	0	0	1
10	1	1	1	0	1

Figure 2.1: A small example of an itemset database, in multiset notation (left) and in binary matrix notation (right)

2.1.1 Frequent Itemset Mining

The problem of frequent itemset mining was proposed by Agrawal et al. [1993]. Given is an itemset database, a database containing a set of *transactions*[2], also called examples. Each transaction consists of a set of *items*. We denote the set of transaction identifiers as $\mathcal{S} = \{1, \ldots, n\}$ and the set of all items in the database as $\mathcal{I} = \{1, \ldots, m\}$. An itemset database $\mathcal{D}$ is represented as a binary matrix of size $n \times m$ with $\mathcal{D}_{ti} \in \{0, 1\}$, or equivalently as a multi-set $\mathcal{D}'$ of itemsets $I \subseteq \mathcal{I}$, such that

$$\mathcal{D}' = \{(t, I) \,|\, t \in \mathcal{S}, I \subseteq \mathcal{I}, \forall i \in I : \mathcal{D}_{ti} = 1\}.$$

A small example of an itemset database is given in Figure 2.1, where for convenience every item is represented as a letter.

There are many databases that can be converted into an itemset database. The traditional example is a supermarket database, in which each transaction corresponds to a customer and every item in the transaction to a product bought by the customer. Attribute-value tables can be converted into an itemset database as well. For categorical data, every attribute-value pair corresponds to an item and every row is converted into a transaction.

The **cover of an itemset** I consists of all transactions in which the itemset occurs. It can be formalized as a function $\varphi_{\mathcal{D}} : 2^{\mathcal{I}} \to 2^{\mathcal{S}}$ which maps an itemset I to a set of transactions as follows:

$$\varphi_{\mathcal{D}}(I) = \{t \in \mathcal{S} \,|\, \forall i \in I : \mathcal{D}_{ti} = 1\}$$

[2]Itemset mining was first applied in a supermarket setting; the terminology still reflects this.

The **frequency of an itemset** I, which is denoted by $frequency_{\mathcal{D}}(I)$, is the size of the cover:

$$frequency_{\mathcal{D}}(I) = |\varphi_{\mathcal{D}}(I)|\,.$$

In the example database we have that the cover $\varphi_{\mathcal{D}}(\{\text{D, E}\}) = \{T_6, T_7\}$ and $frequency_{\mathcal{D}}(\{\text{D,E}\}) = |\{T_6, T_7\}| = 2$. We will omit the $_{\mathcal{D}}$ if the database is clear from the context.

Definition 1 (Frequent Itemset Mining)**.** *Given an itemset database $\mathcal{D}$ and a threshold θ, the frequent itemset mining problem consists of computing the set*

$$\{I \mid I \subseteq \mathcal{I}, frequency_{\mathcal{D}}(I) \geq \theta\}.$$

The threshold θ is called the minimum frequency *threshold. An itemset I with $frequency_{\mathcal{D}}(I) \geq \theta$ is called a* frequent itemset*.*

This coincides with finding

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

where the pattern language is the space of all itemsets, the powerset of $\mathcal{I}$ denoted $\mathcal{P}(\mathcal{I})$, and p specifies the frequency constraint

$$p(I, \mathcal{D}) = true \text{ iff } frequency_{\mathcal{D}}(I) \geq \theta.$$

Note that we are interested in finding *all* itemsets satisfying the frequency constraint. By changing the frequency threshold, an analyst can influence the number of patterns that is returned by the data mining system: the lower the frequency threshold, the larger the number of frequent patterns.

Frequent itemset mining is a challenging problem: given $m = |\mathcal{I}|$ items, the search space contains 2^m possible itemsets. This is illustrated in Figure 2.2 for the example database of Figure 2.1; the frequent itemsets are visualized in a *Hasse diagram*: a line is drawn between two itemsets I_1 and I_2 iff $I_1 \subset I_2$ and $|I_2| = |I_1| + 1$.

Finding the frequent sets by generating all 2^m sets and testing them would take a prohibitive amount of time, even for databases of reasonable size with hundreds of items and thousands of transactions. To overcome this, algorithms have been developed that exploit the subset relation between itemsets to prune away large parts of the search tree.

Search and pruning. The most important property used to prune the search space in traditional algorithms is the *anti-monotonicity* of a function with respect to the subset relation between itemsets.

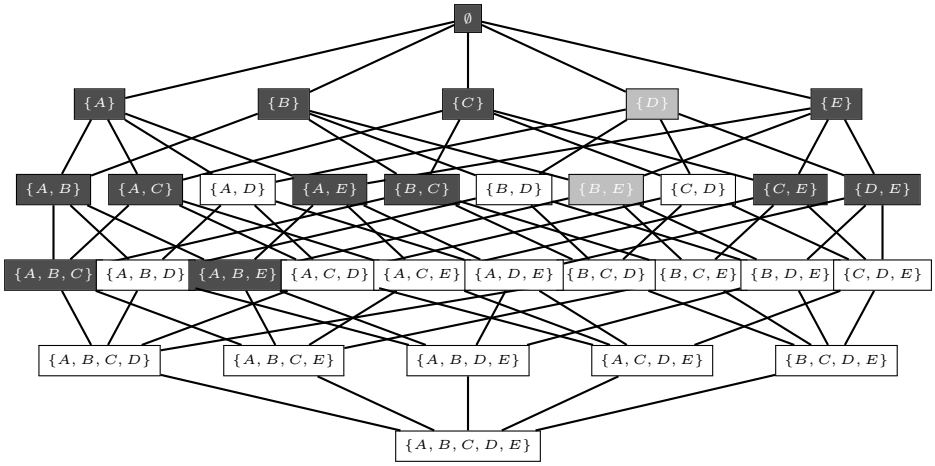


Figure 2.2: A visualization of the search space for the database in Figure 2.1; frequent itemsets for $\theta = 2$ are highlighted. Frequent closed itemsets are highlighted black; non-closed frequent itemsets are grey.

Definition 2 (Anti-Monotonic Constraints)**.** *Assume given two itemsets I_1 and I_2, and a predicate $p(I, \mathcal{D})$ expressing a constraint that itemset I should satisfy on database $\mathcal{D}$. Then the constraint is anti-monotonic iff $\forall I_1 \subseteq I_2 : p(I_2, \mathcal{D}) \implies p(I_1, \mathcal{D})$.*

Indeed, if an itemset I_2 is frequent, any itemset $I_1 \subseteq I_2$ is also frequent, as it must be included in at least the same transactions as I_2. This property allows one to develop search algorithms that do not need to consider all possible itemsets. In case of general to specific search, where one starts from the empty itemset, no itemset $I_2 \supset I_1$ needs to be considered any more once it has been found that I_1 is infrequent.

Starting a search from the empty itemset, there are many ways in which the search space could be traversed, the most important ones being breadth-first search and depth-first search. Initial algorithms for itemset mining were mostly breadth-first search algorithms, of which the Apriori algorithm is the most well-known [Agrawal et al. 1996]. However, more recent algorithms mostly use depth-first search. As we will see later, most CP systems also perform depth-first search. Hence the similarities between CP and depth-first itemset mining algorithms are much larger than between CP and breadth-first mining algorithms. An outline of a general depth-first frequent itemset mining algorithm is given in Algorithm 2.1. The main observations are the following:

- if an item is infrequent in a database, we can remove the corresponding

Algorithm 2.1 Depth-First-Search(Itemset I, Database $\mathcal{D}$)

1: $\mathcal{F} := \{I\}$
2: determine a total order R on the items in $\mathcal{D}$
3: **for all** items i occurring in $\mathcal{D}$ **do**
4: create from $\mathcal{D}$ projected database $\mathcal{D}_i$, containing:
5: - only transactions in $\mathcal{D}$ that contain i
6: - only items in $\mathcal{D}_i$ that are frequent and higher than i in the order R
7: $\mathcal{F} := \mathcal{F} \cup$ Depth-First-Search$(I \cup \{i\}, \mathcal{D}_i)$
8: **end for**
9: return $\mathcal{F}$

column from the database, as no itemset will contain this item and hence the column is redundant.

- once an item is added to an itemset, all transactions not containing this item become irrelevant for the search tree below this itemset; hence we can remove the corresponding row from the database.

The resulting database, which contains a smaller number of transactions having a smaller number of items, is often called a *projected database*. Hence, every time that we add an item to an itemset, we determine which items and transactions become irrelevant, and continue the search for the resulting database, which only contains frequent items and transactions covered by the current itemset. Important benefits are that transactions found to be no longer relevant can be ignored completely, and that the search procedure will never try to add items once they have been found to be infrequent. Thanks to the latter there is no explicit stopping criterion in the algorithm: all items in the projected database (step 3) lead to at least one frequent itemset, namely this item added to I.

Please note that in the projected database, we only include items which are strictly higher in order than the highest item currently in the itemset. The reason for this is that we wish to avoid that the same itemset is found multiple times; for instance, we wish to find itemset $\{A, B\}$ only as a child of itemset $\{A\}$ and not also as a child of itemset $\{B\}$.

An example of a depth-first search tree is given in Figure 2.3, using the same database as in Figure 2.1; we represent the projected database using a list of transaction identifiers per item. The order of the items is assumed to be the alphabetical order. The root is the initial projected database, containing only frequent items (in this case, all items). Each child of the root corresponds to an itemset with one item. Each itemset has a corresponding projected database

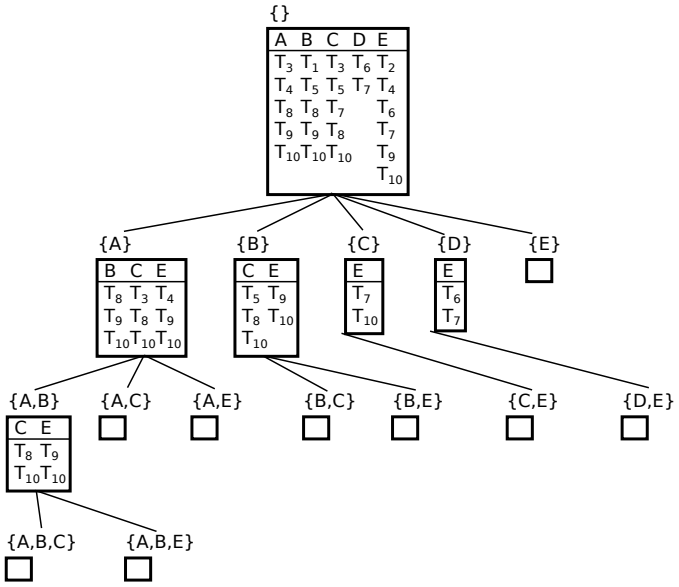


Figure 2.3: The search tree for depth-first frequent itemset miners, for the same example as in Figure 2.1, where the items are ordered alphabetically. Each node consists of an itemset, written in { }, and the corresponding projected database.

containing only frequent items higher than the items chosen so far. For instance, the projected database for itemset $\{C\}$ does not contain D as itemset $\{C, D\}$ has a frequency of 1 and the minimum frequency threshold is 2 in this example. The itemset $\{E\}$ is frequent yet its projected database is empty because all other items are lower in the alphabetic order.

2.1.2 Constraint-based Mining

One of the problems with frequent pattern mining is that the frequency constraint is often not restrictive enough to find interesting patterns. A high frequency threshold has as effect that only well-known pattern are found, which should be avoided. A low threshold on the other hand typically results in a very large number of patterns. Hence using only the frequency constraint results in *too many patterns*, of which few are actually interesting.

Under the umbrella of *constraint-based mining*, many additional types of constraints have been introduced. The general idea is to add constraints

such that the size of $\mathrm{Th}(\mathcal{L}, p, \mathcal{D})$ is reduced. There are three major directions along which the constraint p can be made more restrictive. Using the *anti-monotonicity framework* one can add additional constraints on the individual item- and transaction-sets. *Condensed representations* aim at avoiding redundant itemsets in the solution set. Meanwhile, *discriminative itemsets* are itemsets required to discriminate two databases from each other.

A recurring theme in constraint-based mining is how the constraints in p can be used effectively during search. In a naive scheme, one could simply generate all frequent itemsets and subsequently test the remainder of the constraints. However, the number of frequent itemsets is typically large for low frequency thresholds. Hence, if all constraints can be exploited during search, the search would become a lot more efficient. Doing this is often referred to as *pushing constraints into the mining process*.

Anti-monotonicity framework. The study of how to push constraints in the mining process has led to the creation of the *anti-monotonicity framework* [Bayardo et al. 2000; Pei and Han 2000; Bucila et al. 2003; Bonchi and Lucchese 2007]. This framework tries to exploit the previously defined anti-monotonicity of constraints. For example, like the minimum frequency constraint, a constraint on the maximum size of an itemset is also anti-monotonic: if $|I_2| < \gamma$, for any $I_1 \subseteq I_2$ we know that $|I_1| < \gamma$. A maximum size constraint can be exploited during search in the same way as the minimum frequency constraint; this is true for all constraints that are anti-monotonic.

However, many constraints are not anti-monotonic. Take the minimum size of an itemset: $|I| > \gamma$. This constraint is the reverse of the anti-monotonic maximum size constraint, it is monotonic with respect to set inclusion.

Definition 3 (Monotonic Constraints)**.** *Assume given two itemsets I_1 and I_2, and a predicate $p(I, \mathcal{D})$ expressing a constraint. Then the constraint is monotonic iff $\forall I_1 \subseteq I_2 : p(I_1, \mathcal{D}) \implies p(I_2, \mathcal{D})$.*

Indeed, for the minimum size constraints, if $|I_1| > \gamma$ then for any $I_2 \supseteq I_1$ also $|I_2| > \gamma$.

Monotonic constraints can be exploited during search too. The basic property used is that if I_1 violates the monotonic constraint, all $I_2 \subseteq I_1$ will too. If one would traverse the search tree from specific to general, this property could be used exactly like the anti-monotonic property is when traversing from general to specific [Bucila et al. 2003]. Monotonic constraints can also be exploited by removing transactions that violate it from the (projected) database. If a transaction violates a monotonic constraint, so will any itemset that is a subset

of this transaction, meaning any itemset to whose frequency this transaction could contribute. Hence, this transaction will not change the frequency of the itemsets in the solution set and can be removed [Bonchi and Lucchese 2007]. Monotonic constraints are discussed in more detail in Chapter 3.

Other constraints are neither monotonic nor anti-monotonic, but can be converted to monotonic or anti-monotonic ones. Assume that every item has a weight or a cost c_i, for example, the price of the item. The total cost of an itemset I is then $c(I) = \sum_{i \in I} c(i)$; the minimum average cost constraint can now be defined as $c(I)/|I| \geq \gamma$. This constraint is called *convertible anti-monotone* as we can compute an order on the items in $\mathcal{I}$ such that for any itemset $I \subseteq \mathcal{I}$, every prefix I' of the items in I sorted in this order, also satisfies the constraint. In this case, ordering the items by decreasing cost converts the constraint into an anti-monotonic constraint that can be exploited during search. In general, we can define convertible anti-monotonic constraints as follows:

Definition 4 (Convertible Anti-Monotonic Constraints)**.** *Assume given two itemsets I_1 and I_2, a predicate $p(I, \mathcal{D})$ expressing a constraint, and an order $<$ between items where* $\min(I)$ *and* $\max(I)$ *represent the smallest and largest item in I. Then the constraint is convertible anti-monotonic for this order iff* $\forall I_1 \subseteq I_2, \min(I_2 \setminus I_1) \geq \max(I_1) : p(I_2, \mathcal{D}) \implies p(I_1, \mathcal{D})$.

For many other constraints it has been investigated how to exploit them in an anti-monotonicity framework [Bucila et al. 2003; Bonchi and Lucchese 2007]. We will review such constraints and how they fit in a constraint programming setting in Chapter 3.

A number of efficient systems have been built on the anti-monotonicity framework [Ng et al. 1998; Soulet and Crémilleux 2005; Bonchi and Lucchese 2007]. However, the framework does not solve the constraint-based itemset mining problem in general. For example, convertible anti-monotone constraints require a certain ordering during search. In a problem setting with two such constraints, only one order can be chosen and hence only one constraint will be exploited effectively. The condensed representations explained in the next section do not fit the anti-monotonicity framework either.

Condensed representations Adding constraints on item- and transaction-sets is one way to reduce the number of patterns found. However, a fundamental problem is that many patterns are redundant. An itemset is redundant if its presence in the full solution set can be derived from the other itemsets found. *Condensed representations* [Pasquier et al. 1999; Bayardo 1998; Calders and Goethals 2007] aim at avoiding such redundant itemsets.

Closed itemsets [Pasquier et al. 1999] are a popular form of condensed representation. One way to interpret itemsets is by seeing them as rectangles of ones in a binary matrix. For instance, in our example database of Figure 2.1, itemset {D} is covered by transactions $\{T_6, T_7\}$: $\varphi(\{\text{D}\}) = \{T_6, T_7\}$. The itemset {D} and the transaction set $\{T_6, T_7\}$ select a submatrix which can be seen as a rectangle in the matrix on page 12. Observe that due to the way that we calculate the set of transactions from the set of items, we cannot add a transaction to the set of transactions without including a zero element in the rectangle. However, this is not the case for the columns. In this case, we have $\varphi(\{\text{D}\}) = \varphi(\{\text{D,E}\}) = \{T_6, T_7\}$; we can add item E and still obtain a rectangle containing only ones.

An essential property of a 'maximal' rectangle is that if we consider its transactions, we can derive the corresponding set of items: the largest itemset shared by all transactions must define all columns included in the rectangle. We can introduce a function $\psi_{\mathcal{D}}(T)$ that does so,

$$\psi_{\mathcal{D}}(T) = \{i \in \mathcal{I} \mid \forall t \in T : \mathcal{D}_{ti} = 1\}.$$

Given an itemset I, the itemset $\psi_{\mathcal{D}}(\varphi_{\mathcal{D}}(I))$ is called the *closure* of I.

Definition 5 (Closed Itemset Mining)**.** *Given an itemset database $\mathcal{D}$, the closed itemset mining problem consists of computing the set*

$$\{I \mid I \subseteq \mathcal{I}, I = \psi_{\mathcal{D}}(\varphi_{\mathcal{D}}(I))\}.$$

An itemset I that equals its closure is called a closed itemset*.*

If an itemset is not equal to its closure, this means that we can add an item to the itemset without changing its frequency. Naturally, one can combine closed itemset mining with the frequency constraint to obtain closed frequent itemset mining. All frequent closed itemsets for our example database are highlighted in black in Figure 2.2.

The idea behind closed itemsets has also been studied in other communities; closed itemset mining is in particular related to the problem of finding *formal concepts* in *formal contexts* [Ganter et al. 2005]. Essentially, formal concepts can be thought of as closed itemsets that are found without applying a frequency threshold. In formal concept analysis, the operators φ and ψ are called Galois operators. These operators define a *Galois connection* between the partial orders for itemsets and transaction sets, respectively.

Maximal itemsets [Bayardo 1998; Burdick et al. 2005] are another form of condensed representation. An itemset is called maximal with respect to the frequency constraint, if all supersets of the itemset are infrequent, while all

$\mathcal{D}^+$	Itemset	$\mathcal{D}^-$	Itemset
T_1	{B}	T_2	{E}
T_3	{A,C}	T_4	{A,E}
T_5	{B,C}	T_6	{D,E}
T_8	{A,B,C}	T_7	{C,D,E}
T_{10}	{A,B,C,E}	T_9	{A,B,E}

Figure 2.4: A small positive and negative database, in multiset notation.

subsets are frequent. Hence, such an itemset is maximally frequent compared to its sub- and supersets. The maximal frequent itemset mining problem can be defined as follows.

Definition 6 (Maximal Frequent Itemset Mining)**.** *Given an itemset database $\mathcal{D}$ and a threshold θ, the maximal frequent itemset mining problem consists of computing the set*

$$\{I \mid I \subseteq \mathcal{I}, \mathit{frequency}_{\mathcal{D}}(I) \geq \theta,\ \forall I' \supset I : \mathit{frequency}_{\mathcal{D}}(I') < \theta\}.$$

Maximal frequent itemsets constitute a *border* between itemsets that are frequent and not frequent [Mannila and Toivonen 1997]. They are mostly used in applications where the actual frequency of the itemsets is not of any further importance.

Many other condensed representations have been proposed, differing in the way they define redundancy. They range from exact representations for which the frequency of each pattern can be reconstructed, to lossy representations that discard information regarding frequency [Calders and Goethals 2007]. Examples of other condensed representations include free itemsets [Boulicaut et al. 2000], non-derivable itemsets [Calders and Goethals 2007] and regular itemsets [Ruggieri 2010]. For each of these condensed representations, specialized algorithms have been developed.

Discriminative itemset mining. When two databases are given, one can be interested in finding itemsets that discriminate between these databases. Figure 2.4 shows the database of Figure 2.1 where the transactions are divided between a database $\mathcal{D}^+$ with positive examples and a database $\mathcal{D}^-$ with negative examples. The task is here to find itemsets that allow one to discriminate the transactions belonging to one database from those belonging to the other database. In the example database of Figure 2.4 for instance, the itemset {B,C} only occurs in positive examples, and hence can be said to discriminate positive from negative transactions. Another example is the

itemset {E} which covers all of the negative and only one of the positive transactions.

We will restrict ourself to the case where two databases are given, a positive one and a negative one. This can be extended to multiple databases as well [Nijssen and Kok 2005]. The frequency of an itemset on the positive and negative databases is calculated in the same way as on a single database:

$$\mathit{frequency}_{\mathcal{D}^+}(I) = |\varphi_{\mathcal{D}^+}(I)| \qquad \mathit{frequency}_{\mathcal{D}^-}(I) = |\varphi_{\mathcal{D}^-}(I)| \tag{2.2}$$

Multiple discrimination measures have been proposed that calculate the discriminative power of an itemset in different ways. Discrimination measures are computed over the following *contingency table*, where $p = \mathit{frequency}_{\mathcal{D}^+}(I)$ and $n = \mathit{frequency}_{\mathcal{D}^-}(I)$:

p	$\lvert\mathcal{D}^+\rvert - p$	$\lvert\mathcal{D}^+\rvert$
n	$\lvert\mathcal{D}^-\rvert - n$	$\lvert\mathcal{D}^-\rvert$
$p+n$	$\lvert\mathcal{D}\rvert - (p+n)$	$\lvert\mathcal{D}\rvert$

Given these numbers, we can compute a *discrimination score* $f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|)$. Examples of discrimination measures f include χ^2, information gain, gini index, Fisher score and measures from machine learning such as accuracy, precision and recall (see Chapter 3).

Such discrimination measures can be used to find all discriminating itemsets:

Definition 7 (Discriminative Itemset Mining)**.** *Given two databases $\mathcal{D}^+$ and $\mathcal{D}^-$, a discrimination measure f and a parameter θ, the discriminative itemset mining problem is that of finding all itemsets in*

$$\{I \mid I \subseteq \mathcal{I}, p = \mathit{frequency}_{\mathcal{D}^+}(I), n = \mathit{frequency}_{\mathcal{D}^-}(I), f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|) \geq \theta\}.$$

Whereas we will refer to this problem here as the problem of *discriminative itemset mining* [Morishita and Sese 2000; Cheng et al. 2007; Fan et al. 2008], it is also known under many other names, such as correlated itemset mining [Sese and Morishita 2002; Nijssen and Kok 2005], interesting itemset mining [Bayardo et al. 2000], contrast set mining [Bay and Pazzani 1999], emerging itemset mining [Dong and Li 1999] and subgroup discovery [Wrobel 1997; Kavsek et al. 2003; Grosskreutz et al. 2008]. The problem is also highly related to that of finding a single rule in rule learning [Fürnkranz and Flach 2005]. The key difference is that rule learning in machine learning typically uses heuristic techniques, while in itemset mining typically exhaustive techniques are used to find the optimal itemset.

Constraint-based mining summary. In general, many itemset mining variants have been proposed, including many different constraints. Each new task is typically accompanied by a new algorithm, or variant of an existing algorithm. This makes the combination and reuse of constraints across different tasks hard. The anti-monotonicity framework alleviates this problem in part, but it does not cover the full breath of constraints proposed. Particularly, aspects of constraint-based mining such as condensed representations, discriminative measures and arbitrary combinations of constraints are not well supported.

2.1.3 Pattern Set Mining

Even when using condensed representations or additional constraints, the generated set of patterns is typically still very large. This makes the set of all solutions hard to interpret as the patterns interact with one another. Often, algorithms are used to extract a small set of useful patterns from the set of all solutions. This has led to a typical *step-wise* procedure in which pattern mining only forms an *intermediate* step in the knowledge discovery process. In such an intermediate step, the individual patterns adhering to a set of constraints are exhaustively searched for. These patterns can be called *local* patterns. In the other steps, local patterns are selected and combined in a heuristic way to create a *global* model.

One example is associative classification, where systems such as CBA [Liu et al. 1998] and CMAR [Li et al. 2001] build a classifier from a set of precomputed association rules. Other examples of pattern set mining tasks are concept-learning [Kearns and Vazirani 1994], conceptual clustering [Fisher 1987], redescription mining [Parida and Ramakrishnan 2005] and tiling [Geerts et al. 2004]. For each of these tasks, a small set of patterns is calculated. These tasks are particular instances of the *pattern set mining problem*. This is discussed in more detail in Chapter 5.

The pattern set mining problem can be defined as the problem of finding:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \subseteq \mathcal{L} \mid p(\Pi, \mathcal{D}) \text{ is true}\}, \tag{2.3}$$

where Π is a set of patterns $\Pi = \{\pi_1, \ldots, \pi_s\}$. In its entirety, the pattern set Π forms a global model. The constraint p specifies both local and global constraints at the overall pattern set level.

In a *two step* procedure the constraint p is typically split up into a constraint p_{local} on the local patterns and a constraint p_{global} on the global pattern set. First, all patterns that adhere to the *local constraints* are mined for exhaustively, $\mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D}) = \{\pi \in \mathcal{L} \mid p_{local}(\pi, \mathcal{D}) \text{ is true}\}$. Then, patterns

from $\mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$ are combined under a set of *global constraints*:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \subseteq \mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D}) \,|\, p_{global}(\Pi, \mathcal{D}) \text{ is true}\}. \tag{2.4}$$

Because of the large size of the local pattern set, usually heuristic techniques are used when searching for the global pattern set. The use of exhaustive search has been previously proposed [Zhao et al. 2006; De Raedt and Zimmermann 2007].

In contrast to constraint-based mining, where mining problems are often well formalized and anti-monotonicity is one of the guiding principles for the development of algorithms, no unifying principles or algorithms exist for pattern set mining. The proposed step-wise procedures employ a wide range of heuristics and greedy search strategies. The global constraints p_{global} are not always explicitly specified but hidden in the algorithm. The main problem with these approaches is that it is unclear how they extend towards new and unsolved problems.

A number of approaches to pattern set mining have been explored [Knobbe and Ho 2006; De Raedt and Zimmermann 2007; Khiari et al. 2010]. However, they were either restricted to a small number of settings or did not provide a uniform framework. To the best of our knowledge, there currently is no approach to pattern set mining that can generalize over a large number of tasks.

2.2 Constraint Programming (CP)

In this section we provide a brief summary of the most common approach to constraint programming. More details can be found in text books, notably the "Handbook of Constraint Programming" [Rossi et al. 2006]; we focus on high-level principles and omit implementation issues.

Constraint programming is a declarative paradigm in which the user specifies a problem in terms of its constraints, and the system is responsible for finding solutions that adhere to the constraints. The class of problems that constraint programming systems solve are constraint satisfaction problems and constraint optimisation problems.

Definition 8 (Constraint Satisfaction Problem (CSP))**.** *A CSP* $\mathcal{P} = (\mathcal{V}, D, \mathcal{C})$ *is specified by*

- *a finite set of variables* $\mathcal{V}$*;*
- *an initial domain* D*, which maps every variable* $v \in \mathcal{V}$ *to a set of possible values* $D(v)$*;*

- *a finite set of constraints* $\mathcal{C}$.

A variable $x \in \mathcal{V}$ *is called* fixed *if* $|D(x)| = 1$*; a domain* D *is* fixed *if all its variables are fixed,* $\forall x \in \mathcal{V} : |D(x)| = 1$*. A domain* D' *is called* stronger *than domain* D *if* $D'(x) \subseteq D(x)$ *for all* $x \in \mathcal{V}$*; a domain is* false *if there exists an* $x \in \mathcal{V}$ *such that* $D(x) = \emptyset$*. A* constraint $C(x_1, \ldots, x_k) \in \mathcal{C}$ *is relation between variables* $\{x_1, \ldots, x_k\} \subseteq \mathcal{V}$ *that evaluates to either true or false.*

A solution to a CSP is a fixed domain D' *stronger than the initial domain* D *that satisfies all constraints. Abusing notation for a fixed domain, we must have that* $\forall C(x_1, \ldots, x_k) \in \mathcal{C} : C(D'(x_1), \ldots, D'(x_k)) = true$

A *constraint optimisation problem* is a CSP $(\mathcal{V}, D, \mathcal{C})$ augmented with an objective function that maps the variables of $\mathcal{V}$ to an evaluation score v_s. The problem is then to find the solution that satisfies all constraints and is maximal (or minimal) with respect to the objective function.

A distinguishing feature of CP is that it does not focus on a specific set of constraint types. Instead it provides general principles for solving problems with any type of variable or constraint. This sets it apart from integer linear programming (ILP), which focuses on linear constraints on integer variables, and from propositional satisfiability (SAT) solving, which solely uses Boolean variables and clause constraints.

Example 1. *Assume we have four people that we want to allocate to two offices, and that every person has a list of people that he does not want to share an office with. Furthermore, every person has identified rooms he does not want to occupy. We can represent an instance of this problem with four variables which represent the persons, and inequality constraints which encode the room-sharing constraints:*

$$D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{1, 2\}$$
$$\mathcal{C} = \{x_1 \neq 2, x_1 \neq x_2, x_3 \neq x_4\}.$$

In the above example of a problem specification, we had to make some modeling decisions. For example, we could have modeled each of the offices as a set variable describing the set of people in that office. When modeling a problem it is always possible to specify it in different ways. As different constraints are implemented in different ways, the modeling choices made can have an effect on runtime efficiency.

Algorithm 2.2 Constraint-Search(D)

```
1:  D :=propagate(D)
2:  if D is a false domain then
3:      return
4:  end if
5:  if ∃x ∈ V : |D(x)| > 1 then
6:      x := arg min_{x∈V, D(x)>1} f(x)
7:      D_s :=split(D(x)))
8:      Constraint-Search(D ∪ {x ↦ D_s})
9:      Constraint-Search(D ∪ {x ↦ D(x) \ D_s})
10: else
11:     Output solution
12: end if
```

2.2.1 Search and propagation

The task of a solver is to find the solutions that satisfy the constraint specification.

The simplest algorithm to solve CSPs enumerates all possible fixed domains, and checks all constraints on each of these domains; clearly this approach is inefficient. Constraint solvers perform a more intelligent search in which search, assigning values to variables, is interleaved with propagation, the act of removing inconsistent values from the domain using constraints. By removing inconsistent values during search, the number of search steps required can be drastically reduced. We review each of those principles in turn.

Search. Many CP systems perform a type of depth-first search, as given in Algorithm 2.2 [Schulte and Stuckey 2008; Rossi et al. 2006]. Other search strategies have been investigated as well [Van Hentenryck et al. 2000; Perron 1999], but we restrict ourselves here to the most common case.

The search tree of a CSP is ordered from general to specific domains. The root node consists of the initial domain D containing all the possible values of each variable. Solutions are found in the leaves of the search tree, where every variable v has only one value in its domain $D(v)$. In each node of the search tree the algorithm branches by splitting the domain of a variable (line 7 in Algorithm 2.2); for Boolean variables, $\text{split}(\{0,1\}) = \{0\}$ or $\text{split}(\{0,1\}) = \{1\}$. The search backtracks when a violation of a constraint is found (line 2). The search is further optimized by carefully choosing the variable that is fixed next

(line 6); here a function $f(x)$ ranks variables, for instance, by determining which variable is involved in the highest number of constraints.

For optimisation problems, the above algorithm can easily be changed into a branch-and-bound algorithm. In this case a constraint is added on the evaluation score v_s; this constraint is updated each time a better solution is found than the currently best known one (line 11), and hence avoids searching for solutions worse than that one.

Propagation. The main concept used to speed up the search is constraint propagation (line 1). Propagation reduces the domain of variables such that the domain remains *locally consistent*. A domain is (globally) consistent if all possible combinations of value assignments satisfy the constraints. This can be hard to determine and even impossible to maintain during search. Hence a form of local consistency is maintained in which only one or a subset of constraints are considered. In a locally consistent domain, those values for which a constraint can determine that they can never be part of a solution are removed. The main motivation for maintaining consistencies is to ensure that the backtracking search does not unnecessarily branch over such values, thereby significantly speeding up the search.

Many types of local consistencies have been defined for constraint networks, such as node consistency, arc consistency and path consistency; see [Bessiere 2006] for an overview. In the context of Algorithm 2.2, we only consider local consistency methods that operate on individual constraints.

To maintain local consistencies for individual constraints, *propagators* or propagation rules are used. Each constraint is implemented by a propagator. A propagator takes the domain as input and outputs a failed domain in case it can determine that the constraint can no longer be satisfied, i.e. if there exists no

$$\text{fixed } D' \text{ stronger than } D \text{ with } C(D'(x_1), \ldots, D'(x_k)) = \text{true}. \tag{2.5}$$

When possible, the propagator will remove values from the domain that can never satisfy the constraint, giving as output a stronger, locally consistent domain. More formally, a value c should be removed from the domain of a variable $\tilde{x}$ if there is no

$$\text{fixed } D' \text{ stronger than } D \text{ with } D'(\tilde{x}) = c \text{ and } C(D'(x_1), \ldots, D'(x_k)) = \text{true}. \tag{2.6}$$

This type of constraint propagation ensures *domain consistency*: every value in the domain is consistent with the constraint at hand.

Algorithm 2.3 Conceptual propagator for $x_1 \neq x_2$

1: **if** $|D(x_1)| = 1$ **then**
2: $\quad D(x_2) = D(x_2) \setminus D(x_1)$
3: **end if**
4: **if** $|D(x_2)| = 1$ **then**
5: $\quad D(x_1) = D(x_1) \setminus D(x_2)$
6: **end if**

Consider the constraint $x_1 \neq x_2$, for which the corresponding propagator is given in Algorithm 2.3. The propagator can only propagate when x_1 or x_2 is fixed (lines 1 and 4). If one of them is, its value is removed from the domain of the other variable (lines 2 and 5). In this propagator there is no need to explicitly check whether the constraint is violated, as a violation results in an empty and thus false domain in line 2.

The repeated application of propagators can lead to increasingly stronger domains. Propagators are repeatedly applied until a *fixed point* is reached in which the domain does not change any more.

Example 2 (Example 1 continued)**.** *The initial domain of this problem is not consistent: the constraint* $x_1 \neq 2$ *cannot be satisfied when* $D(x_1) = \{2\}$ *so value* 2 *is removed from the domain of* x_1*. Subsequently, the propagator for the constraint* $x_1 \neq x_2$ *is activated, which removes value* 1 *from* $D(x_2)$*. At this point, we obtain a fixed point with* $D(x_1) = \{1\}$*,* $D(x_2) = \{2\}$*,* $D(x_3) = D(x_4) = \{1, 2\}$*. Person 1 and 2 have now each been allocated to an office, and two rooms are possible for person 3 and 4. The search branches over* x_3 *and for each branch, constraint* $x_3 \neq x_4$ *is propagated; a fixed point is then reached in which every variable is fixed, and a solution is found.*

Maintaining domain consistency can be computationally expensive as for every constraint, potentially every value in the domain has to be verified. A weaker form of local consistency is called *bound consistency*. In this case, only the lower- or upper-bound of a variable's domain are changed. A propagator will try to narrow the domain of a variable to that range of values for which it still believes a solution can be found, but does not maintain consistency for individual values. For variables with Boolean domain, i.e. $D(x) = \{0, 1\}$, there is no difference between bound and domain consistency.

Bound consistency can straightforwardly be applied on constraints of the form $Function(V) \lesseqgtr X$ where $\lesseqgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$ and V is a set of variables and X is an integer variable or a constant. The upper-bound of a variable V_i is the largest value in its domain, which we denote by $Upperbound(V_i) = \max D(V_i)$; likewise the lower-bound is the minimal value in the domain. Given the domain

of a set of variables V, the upper- and lower-bound on the outcomes of a function $f(V)$ can often be calculated using the upper- and lower-bounds of the variables in V. The function f does not need to be monotonic, though the bounds of a function can often be calculated by exploiting the monotonicity of the operators composing the function. This is illustrated in the table below for basic arithmetic functions over 2 variables:

Function	Lower-bound	Upper-bound
$V_1 + V_2$	$\min D(V_1) + \min D(V_2)$	$\max D(V_1) + \max D(V_2)$
$V_1 - V_2$	$\min D(V_1) - \max D(V_2)$	$\max D(V_1) - \min D(V_2)$
$V_1 * V_2$	$\min D(V_1) * \min D(V_2)$	$\max D(V_1) * \max D(V_2)$
$V_1 \div V_2$	$\min D(V_1) \div \max D(V_2)$	$\max D(V_1) \div \min D(V_2)$

These bounds can be used to update the domain of the variables. In a constraint of the kind $Function(V) \lesseqgtr X$ where X is a variable, we can update the bounds of the variable X. Furthermore we can iteratively isolate one of the variables in V and use the same mechanism to update that variable's bounds.

In the next section, we investigate the propagators of two constraints in more detail.

2.2.2 Summation and reified summation constraints

We will make extensive use of two types of constraints over Boolean variables, namely the *summation constraint*, equation (2.7), and *reified summation constraint*, equation (2.10).

Summation constraint Given a set of variables $V \subseteq \mathcal{V}$, weights w_x for each variable $x \in V$ and a constant θ, the general form of the summation constraint is:

$$\sum_{x \in V} w_x x \geq \theta. \tag{2.7}$$

The first task of the propagator is to discover as early as possible whether the constraint is violated. To this aim, the propagator needs to determine whether the upper-bound of the sum is still above the required threshold; filling this constraint in equation 2.5, we need to check whether:

$$\max_{\text{fixed } D' \text{ stronger than } D} \left(\sum_{x \in V} w_x D'(x) \right) \geq \theta. \tag{2.8}$$

A more intelligent method to evaluate this property works as follows. We denote the set of variables with a positive, respectively negative, weight by $V^+ = \{x \in V | w_x \geq 0\}$ and $V^- = \{x \in V | w_x < 0\}$, the bounds of the sum are now defined as:

$$\max(\sum_{x \in V} w_x x) = \sum_{x \in V^+} w_x * (\max D(x)) + \sum_{x \in V^-} w_x * (\min D(x))$$

$$\min(\sum_{x \in V} w_x x) = \sum_{x \in V^+} w_x * (\min D(x)) + \sum_{x \in V^-} w_x * (\max D(x))$$

These bounds allow one to determine whether an inequality constraint $\sum_{x \in V} w_x x \geq \theta$ can still be satisfied.

The second task of the propagator is to maintain the bounds of the variables in the constraint, which in this case are the variables in V. In general, for every variable $\tilde{x} \in V$, we need to update $D(\tilde{x})$ such that $\min D(\tilde{x})$ is the lowest value c for which there exists a domain D' with

$$\text{fixed } D' \text{ stronger than } D,\ D'(\tilde{x}) = c \text{ and } \left(\sum_{x \in V} w_x D'(x)\right) \geq \theta. \tag{2.9}$$

Also this can be computed efficiently; essentially, for Boolean variables $\tilde{x} \in V$ we can update all domains as follows:

- $D(\tilde{x}) \leftarrow D(\tilde{x}) \setminus \{0\}$ if $w_{\tilde{x}} \in V^+$ and $\max(\sum_{x \in V} w_x x) - w_{\tilde{x}} < \theta$;
- $D(\tilde{x}) \leftarrow D(\tilde{x}) \setminus \{1\}$ if $w_{\tilde{x}} \in V^-$ and $\theta \leq \max(\sum_{x \in V} w_x x) < \theta - w_{\tilde{x}}$.

Example 3. *Let us illustrate the propagation of the summation constraint. Given*

$$D(x_1) = \{1\}, D(x_2) = D(x_3) = \{0, 1\},$$
$$2 * x_1 + 4 * x_2 + 8 * x_3 \geq 3;$$

we know that at least one of x_2 and x_3 must have value 1, but we cannot conclude that either one of these variables is certainly zero or one. The propagator does not change any domains. On the other hand, given

$$D(x_1) = \{1\}, D(x_2) = D(x_3) = \{0, 1\},$$
$$2 * x_1 + 4 * x_2 + 8 * x_3 \geq 7;$$

the propagator determines that the constraint can never be satisfied if x_3 is false, so $D(x_3) = \{1\}$.

Reified summation constraint The second type of constraint we will use extensively is the reified summation constraint. Reified constraints are a common construct in constraint programming [Van Hentenryck and Deville 1993; Van Hentenryck et al. 1998]. Essentially, a reified constraint binds the truth value of a constraint C' to a Boolean variable b:

$$b \leftrightarrow C'.$$

In principle, C' can be any constraint. In this thesis, C will usually be a constraint on a sum. In this case we speak of a *reified summation constraint*:

$$b \leftrightarrow \sum_{x \in V} w_x x \geq \theta \tag{2.10}$$

This constraint states that b is true if and only if the weighted sum of the variables in V is higher than θ. Essential is the propagation to the domain of variable b, which is updated as follows:

- $D(b) \leftarrow D(b) \setminus \{1\}$ if $max(\sum_{x \in V} w_x x) < \theta$;
- $D(b) \leftarrow D(b) \setminus \{0\}$ if $min(\sum_{x \in V} w_x x) \geq \theta$.

In addition, in some constraint programming systems, constraint propagators can also rewrite constraints to simpler forms, for example by removing constant elements. In case of the reified summation constraint, if $D(b) = \{1\}$, the reified constraint can be rewritten to the constraint $\sum_{x \in V} w_x x \geq \theta$; if $D(b) = \{0\}$, the constraint can be rewritten as $\sum_{x \in V} w_x x < \theta$.

2.2.3 Detailed example

In this section we work out a detailed example including a problem specification with reified constraints. We also provide a step-by-step explanation of the resulting search-propagation interaction that would happen in a solver.

Example 4. *A family of 5, consisting of a mother, a father, a grandmother and two children has won a holiday for 3 people. The parents decide that at least one of them should join, but the father will only join iff either the mother or the grandmother, but not both, goes. We can model this problem as a CSP by having a variable for every family member, namely G (grandmother), F (father), M (mother) and Ch_1 and Ch_2 (the children). If a person joins, the corresponding variable has value* 1 *and* 0 *otherwise; the domain of every variable is* $\{0, 1\}$*, line (2.11) below. We will specify the constraints by summing over these Boolean variables. In line (2.12) below, we specify that at least one*

of the parents has to join. Line (2.13) specifies that the father joins iff either the mother or grandmother joins, using the shorthand notation that F is true iff $F \neq 0$. Finally, line (2.14) specifies that only 3 people can go on the holiday, by constraining the sum of all the variables to be equal to 3.

$$D(G), D(F), D(M), D(Ch_1), D(Ch_2) = \{0, 1\} \tag{2.11}$$

$$F + M \geq 1 \tag{2.12}$$

$$F \leftrightarrow M + G = 1 \tag{2.13}$$

$$G + F + M + Ch_1 + Ch_2 = 3 \tag{2.14}$$

Let us illustrate how the search and propagation are performed for this example. We will abbreviate the domain value $\{0\}$ to 0, $\{1\}$ to 1 and $\{0, 1\}$ to ? such that we can write the initial domain of the variables $\langle G, F, M, Ch_1, Ch_2\rangle$ as $\langle ?, ?, ?, ?, ?\rangle$. This initial domain is depicted in the root of the search tree in Figure 2.5.

Initially none of the constraints can be propagated, so the search will pick a variable and assign a value to it. Which variable and value to pick can be defined by so-called variable and value order heuristics. The choice of the heuristics can have a huge impact on runtime efficiency, as different choices will lead to differently shaped search trees. We will use a general variable ordering heuristic that is known to often perform well, namely to dynamically choose the variable that occurs in the most constraints. As value ordering we will first consider exclusion ($V = 0$) followed by inclusion ($V = 1$). In our example, initially the variables F and M both appear in 3 constraints, so the first variable of these two would be chosen and set to $F = 0$. This leads to the search node with domain values $\langle ?, 0, ?, ?, ?\rangle$, as shown in the upper left of Figure 2.5.

Because of $F = 0$, constraint $F + M \geq 1$ propagates that the mother has to join on the holiday trip ($M = 1$). In constraint $F \leftrightarrow M + G = 1$ the reified variable F is false, so the inverse constraint is posted: $M + G \neq 1$. Because of $M = 1$, this constraint propagates $M + G \neq 1 \Rightarrow 1 + G \neq 1 \Rightarrow G \neq 0$, so $G = 1$. At this point, the domain values are $\langle 1, 0, 1, ?, ?\rangle$. Constraint $G+F+M+Ch_1+Ch_2 = 3$ can now be simplified: $1 + 0 + 1 + Ch_1 + Ch_2 = 3 \Rightarrow Ch_1 + Ch_2 = 1$. This constraint cannot be simplified any further, so our partial solution remains $\langle 1, 0, 1, ?, ?\rangle$. The search now branches over $Ch_1 = 0$, which leads constraint $Ch_1 + Ch_2 = 1$ to propagate that $Ch_2 = 1$. This results in the first solution: $\langle 1, 0, 1, 0, 1\rangle$. The search backtracks to $\langle 1, 0, 1, ?, ?\rangle$ and branches over $Ch_1 = 1$, leading to the solution $\langle 1, 0, 1, 1, 0\rangle$. The search backtracks to $\langle ?, ?, ?, ?, ?\rangle$ and branches over $F = 1$. Constraint $F + M \geq 1$ is now satisfied, so it is removed from consideration. In constraint $F \leftrightarrow M + G = 1$ the reified variable F is

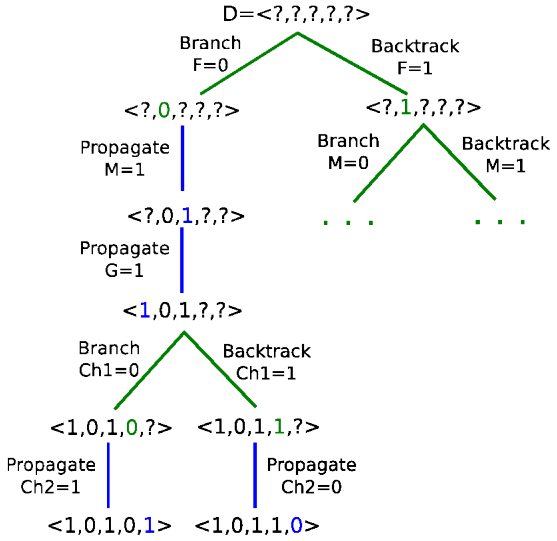


Figure 2.5: Search tree for the holiday example.

true, so the reified constraint is replaced by $M + G = 1$. This does not lead to any further propagation. Constraint $G + F + M + Ch_1 + Ch_2 = 3$ is simplified to $G + M + Ch_1 + Ch_2 = 2$, but remains bound-consistent. Since no more propagation can happen, the search procedure chooses the most constrained variable M and sets $M = 0$: $\langle ?, 1, 0, ?, ? \rangle$. The CP solver will continue to alternate propagation and search in this way, until all solutions are found.

2.2.4 Constraint solvers

Many constraint solving algorithms have been proposed in the literature [Rossi et al. 2006] and a large number of software packages for constraint solving exist [Schulte and Carlsson 2006]. We here review some of the design decisions that differentiate constraint solving methods.

Host language. The most obvious difference between solvers is the host language. Traditionally, Constraint Logic Programming solvers integrate constraint solving in a logic programming environment such as Prolog. Example solvers include SICStus Prolog [Carlsson et al. 1997], GNU Prolog [Diaz and Codognet 2001] and ECLiPSe [Apt and Wallace 2007]. An alternative is to implement a constraint solver as a library in an imperative

language. Example solvers include Ilog Solver (C++, [ILOG S.A. 2003]), Gecode (C++ [Gecode Team 2010]), Choco (Java, [Laburthe 2000]) and Jacop (Java, [Kuchcinski and Szymanek 2011]). When implementing a CP solver in Prolog much of the backtracking infrastructure can be reused, while in an imperative language there is more freedom for optimisation in the low-level mechanisms.

Variable types. Constraint programming enforces no real restriction on the type of variables that can be used. In practice, most research has focussed on variable types with finite domains, more specifically finite domain integers. Other variable types have been investigated too, such as reals and finite domain graphs. In addition to finite domain integers, modern solvers often also support Booleans and finite integer sets.

Domains of variables can be implemented in different ways, for example, representing an integer by its bounds or by its full domain, and implementing a full domain by a list of intervals or with a bitvector. Each choice has its advantages and disadvantages, depending on the properties of the CSP one is trying to solve.

Search states. When a domain fails the search procedure will backtrack. Hence, the state of the nodes in the search tree needs to be maintained in such a way that backtracking is possible. A number of different choices are possible, namely trailing, recomputation and copying. Trailing, the most often used method, maintains for every node in the search tree a trail of changes that happened (e.g., a domain change). A state is restored by undoing these changes. In recomputation, only the variable assignment corresponding to each branch is saved. A state is restored by re-doing the variable assignments that lead to that node; including the propagation that stems from those assignments. Copying on the other hand copies the entire state, and restoration is done by restoring that state as 'active' state. Each of the choices has a memory consumption versus computation time trade-off [Reischuk et al. 2009].

Constraint activation. When the domain of a variable changes, constraints might be able to propagate that information and hence need to be activated. In modern solvers, propagators are activated using propagation events [Schulte and Carlsson 2006]. Events can be used at different levels of granularity. In a coarse-grained setting, when a domain changes each of the constraints containing that variable is notified. In a fine-grained setting, each constraint is notified of the variable that changed as well as how it changed. Propagators can use events to avoid unnecessary computations. For example, a bound consistent

propagator should only recompute the bounds if the minimum or maximum value of one of its variables changed. Propagation events have their roots in the arc consistency algorithm AC-5 [Van Hentenryck et al. 1992]. Fine-grained propagation avoids unnecessary activation and computation of constraints, but requires more administrative overhead for the solver.

Each of the above choices have their advantages and disadvantages. It is therefore unsurprising that there exist a multitude of solvers that differ in many ways. Whether a choice is an advantage or not often depends on the application and problems at hand. In this thesis, we make extensive use of the Gecode system [Gecode Team 2010]. In the following section we describe some of its design decisions and motivate its use in this thesis.

2.2.5 The Gecode solver

Its authors describe Gecode[3] as a constraint solver with state-of-the-art performance while being modular and extensible. We review the choices made in Gecode for the design decisions described above:

Host language Gecode is implemented in C++ and great care is taken to make it efficient. It is one of the fastest general-purpose CP system and has won all MiniZinc solver challenges [Stuckey 2011] so far (2008-2011).

Variable types Gecode supports finite domain Booleans, integers and sets. The domain of integers is implemented as a list of intervals, each interval consisting of a minimum and a maximum value. The domain of sets is represented by a lower- and upper-bound set augmented with cardinality information. The lower- and upper-bound sets are implemented as lists of intervals.

Search states Gecode uses a hybrid copying/recomputation strategy in which the search tree is copied during search, after visiting a predefined number of nodes. Intermediate nodes are restored by recomputation starting from the most recently copied node available.

Constraint activation Gecode uses coarse-grained activation in which a propagator is activated if at least one of its variables changes. Additionally, it provides events indicating whether the changes involved the bounds of variables, a value in the domain, or whether a variable was fixed. It also supports a fine-grained setting using propagator *advisors* that are activated with information on which variable changed, and how

[3] http://www.gecode.org

it changed. While in the coarse-grained setting, propagators often have to recompute their global state, fine-grained advisors allow for incremental propagation.

When a solver is a library in an efficient (imperative) language, input-output operations such as reading files and writing all solutions can be written in the same efficient language. For data-intensive applications such as data mining, such low-level details can make a difference in the practical usability of a solver.

Gecode's strong points are that it is not only efficient, but also open and extendible. The software is written and documented in such a way that anybody can add custom constraints and propagators and even variable types to the system. This makes it a useful research platform for data mining. The ability to add constraints will be essential when dealing with domain-specific constraints such as correlation in Chapter 3 and proximity of genes in Chapter 7.

In general, the use of constraint programming for data mining in this thesis is not only possible because of the principled foundations of search and propagation, but also because of the public availability of efficient and easy to use solvers.

Chapter 3

Constraint-based Itemset Mining using CP

This chapter[1] introduces the *constraint programming for itemset mining* framework that forms the basis of this thesis. We show how typical constraint-based mining tasks can be formulated in CP, and how the framework compares to traditional itemset mining algorithms. We make extensive use of existing constraint solvers while the next chapter investigates a new solver specifically designed for itemset mining.

3.1 Introduction

Itemset mining is probably the best studied problem in the data mining literature. The introduction of a wide variety of constraints and algorithms for solving *constraint-based itemset mining* problems has enabled the application of itemset mining to numerous problems, ranging from web mining to bioinformatics [Han et al. 2007]. This progress has resulted in many effective and scalable itemset mining systems and algorithms, usually optimized to specific tasks and constraints. This algorithmic focus can make it non-trivial to extend such systems to accommodate new constraints or combinations thereof.

[1] Based on the journal paper "Itemset mining: A constraint programming perspective" [Guns et al. 2011b], which in turn is based on conference papers [De Raedt et al. 2008; Nijssen et al. 2009; De Raedt et al. 2010].

The question that arises in this context is whether the principles of constraint programming can also be applied to itemset mining. This approach would specify data mining models using general and declarative constraint satisfaction primitives; this should make it easy to incorporate new constraints and combinations thereof as – in principle – only the model needs to be extended to specify the problem and general purpose solvers can be used for computing solutions.

The contribution of this chapter is that we answer the above question positively by showing that the general, off-the-shelf constraint programming methodology can indeed be applied to the specific problems of constraint-based itemset mining. We show how a wide variety of itemset mining problems (such as frequent, closed and cost-based) can be modeled in a constraint programming language and that general purpose out-of-the-box constraint programming systems can effectively deal with these problems.

While frequent, closed and cost-based itemset mining are ideal cases, for which the existing constraint programming modeling language used suffices to tackle the problems, this cannot be expected in all cases. Indeed, in our formulation of discriminative itemset mining, we introduce a novel primitive by means of a *global constraint*. This is common practice in constraint programming, and the identification and study of global constraints that can effectively solve specific subproblems has become a branch of research on its own [Beldiceanu et al. 2007]. Here, we have exploited the ability of constraint programming to serve as an integration platform, allowing for the free combination of new primitives with existing ones. This property allows to find closed *discriminative* itemsets effectively, as well as discriminative patterns adhering to any other constraint(s). Furthermore, casting the problem within a constraint programming setting also provides us with new insights in how to solve discriminative pattern mining problems. These insights lead to important performance improvements over state-of-the-art discriminative data mining systems.

A final contribution is that we compare the resulting declarative constraint programming framework to well-known state-of-the-art algorithms in data mining. It should be noted that any such comparison is difficult to perform; this already holds when comparing different data mining (resp. constraint programming) systems to one another. In our comparison we focus on high-level concepts rather than on specific implementation issues. Nevertheless, we demonstrate the feasibility of our approach using our CP4IM implementation that employs the state-of-the-art constraint programming library Gecode [Schulte and Stuckey 2008], which was developed for solving general constraint satisfaction problems. While our analysis reveals some weaknesses when applying this particular library to some itemset mining problem, it also reveals that Gecode can already outperform state-of-the-art

data mining systems on some tasks.

The chapter is organized as follows. Section 3.2 revisits the basic problem of frequent itemset mining and discusses how this problem can be addressed using constraint programming techniques. The following sections then show how alternative itemset mining constraints and problems can be dealt with using constraint programming: Section 3.3 studies closed itemset mining, Section 3.4 considers discriminative itemset mining, and Section 3.5 shows that the typical monotonicity-based problems studied in the literature can also be addressed in the constraint programming framework. We also study in these sections how the search of the constraint programming approach compares to that of the more specialized approaches. The CP4IM approach is then evaluated in Section 3.6, which provides an overview of the choices made when modeling frequent itemset mining in a concrete constraint programming system and compares the performance of this constraint programming system to specialized data mining systems. Finally, Section 3.7 concludes.

3.2 Frequent Itemset Mining

We start with the problem of frequent itemset mining, which we introduced in Section 2.1.1 on page 12. We formulate two CP models for this case; the difference between the initial model and the improved one is that the latter uses *reified* constraints. The improved model yields better propagation as shown by an analysis of the resulting search-propagation interactions.

3.2.1 Initial Constraint Programming Model

We formalized the problem of frequent itemset mining in Definition 1 as

$$\{I \mid I \subseteq \mathcal{I}, \mathit{frequency}_{\mathcal{D}}(I) \geq \theta\},$$

where $\mathit{frequency}_{\mathcal{D}}(I) = |\varphi_{\mathcal{D}}(I)|$ and θ is the minimum frequency threshold. We can make the set of transactions covered by the itemset explicit: $T = \varphi_{\mathcal{D}}(I)$. This results in the following equivalent problem formulation:

$$\{(I,T) \mid I \subseteq \mathcal{I}, T \subseteq \mathcal{S}, T = \varphi_{\mathcal{D}}(I), |T| \geq \theta\}, \tag{3.1}$$

This yields the same solutions as the original problem because the set of transactions T is completely determined by the itemset I. We use this observation to model the frequent itemset mining problem in constraint programming by modeling $T = \varphi_{\mathcal{D}}(I)$ and $|T| \geq \theta$ as two separate constraints.

We will refer to $T = \varphi_{\mathcal{D}}(I)$ as the *coverage constraint* while $|T| \geq \theta$ is a *frequency constraint*.

To model this formalization in CP, we need to represent the set of items I and the set of transactions T. In our model we use a boolean variable I_i for every individual item i; furthermore we use a boolean variable T_t for every transaction t. An itemset I is represented by setting $I_i = 1$ for all $i \in I$ and $I_i = 0$ for all $i \notin I$. The variables T_t represent the transactions that are covered by the itemset, i.e. $T = \varphi(I)$; $T_t = 1$ iff $t \in \varphi(I)$. One assignment of values to all I_i and T_t corresponds to one itemset and its corresponding transaction set.

We now show that the coverage constraint can be formulated as follows.

Property 1 (Coverage Constraint)**.** *Given a database $\mathcal{D}$, an itemset I and a transaction set T, then*

$$T = \varphi_{\mathcal{D}}(I) \iff \big(\forall t \in \mathcal{S} : \quad T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0\big), \tag{3.2}$$

or equivalently,

$$T = \varphi_{\mathcal{D}}(I) \iff \big(\forall t \in \mathcal{S} : \quad T_t = 1 \leftrightarrow \bigwedge_{i \in \mathcal{I}} (\mathcal{D}_{ti} = 1 \vee I_i = 0)\big), \tag{3.3}$$

where $I_i, T_t \in \{0, 1\}$ and $I_i = 1$ iff $i \in I$ and $T_t = 1$ iff $t \in T$.

Proof. Essentially, the constraint states that for one transaction t, all items i should either be included in the transaction ($\mathcal{D}_{ti} = 1$) or not be included in the itemset ($I_i = 0$):

$$\begin{aligned}
& T = \varphi_{\mathcal{D}}(I) = \{t \in \mathcal{S} | \forall i \in I : \mathcal{D}_{ti} = 1\} \\
\iff & \forall t \in \mathcal{S} : \quad T_t = 1 \leftrightarrow \forall i \in I : \mathcal{D}_{ti} = 1 \\
\iff & \forall t \in \mathcal{S} : \quad T_t = 1 \leftrightarrow \forall i \in I : 1 - \mathcal{D}_{ti} = 0. \\
\iff & \forall t \in \mathcal{S} : \quad T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0.
\end{aligned}$$

The representation as a clause in equation (3.3) follows from this. □

It is quite common in constraint programming to encounter different ways to model the same problem or even the same conceptual constraint, as above. How propagation is implemented for these constraints can change from solver to solver. For example, *watched literals* [Gent et al. 2006b], could be used for

Algorithm 3.1 CP model of frequent itemset mining, in Essence'

```
1: given NrT, NrI : int
2: given TDB : matrix indexed by [int(1..NrT),int(1..NrI)] of int(0..1)
3: given Freq : int
4: find Items : matrix indexed by [int(1..NrI)] of bool
5: find Trans : matrix indexed by [int(1..NrT)] of bool
6: such that
7: $ Coverage constraint (Equation (3.2))
8: forall t: int(1..NrT).
9:   Trans[t] <=> ((sum i: int(1..NrI). (1-TDB[t,i])*Items[i]) <= 0),
10: $ Frequency constraint (Equation (3.4))
11: (sum t: int(1..NrT). Trans[t]) >= Freq.
```

the clause constraint, leading to different runtime and memory characteristics compared to a setting where no watched literals are used. We refer to the appendix section at the end of this chapter for an in-depth study of such characteristics.

With the coverage constraint, a transaction variable will only be true if the corresponding transaction covers the itemset. Counting the frequency of the itemset can now be achieved by counting the number of transactions for which $T_t = 1$.

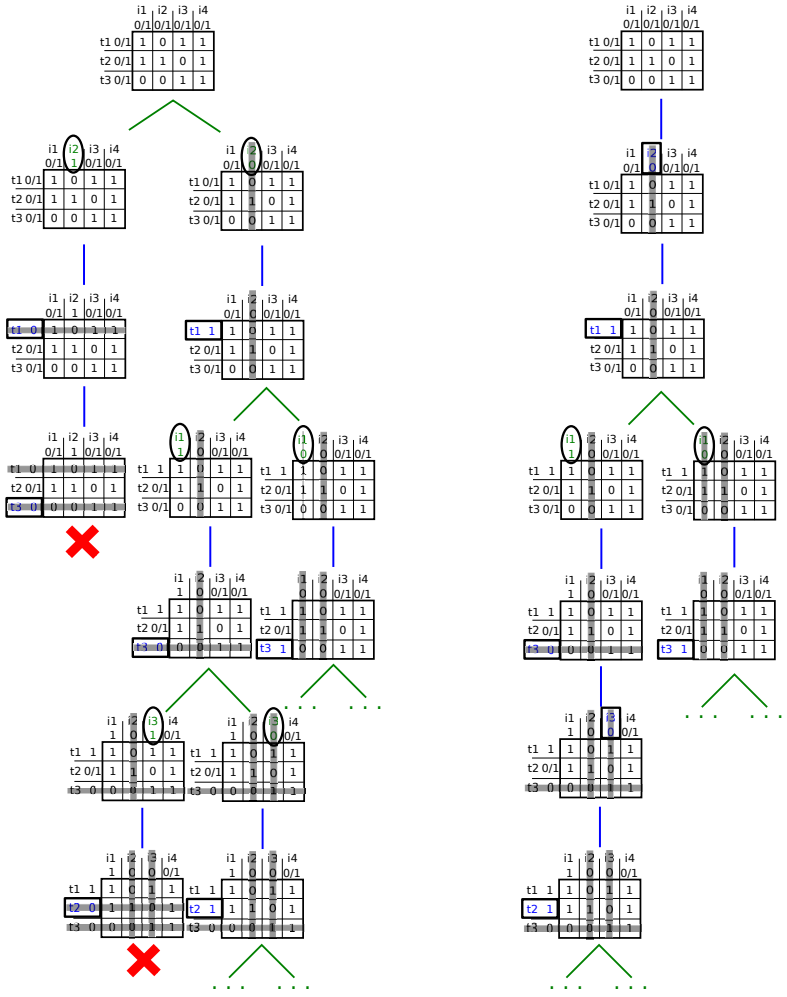
Property 2 (Frequency Constraint)**.** *Given a database $\mathcal{D}$, a transaction set T and a threshold θ, then*

$$|T| \geq \theta \iff \sum_{t \in \mathcal{S}} T_t \geq \theta, \tag{3.4}$$

where $T_t \in \{0, 1\}$ and $T_t = 1$ iff $t \in T$.

We can now model the frequent itemset mining problem as a combination of the coverage constraint (3.2) and the frequency constraint (3.4). To illustrate this, we provide an example of our model in the Essence' language in Algorithm 3.1. Essence' is a solver-independent modeling language; it was developed to support intuitive modeling, abstracting away from the underlying solver technology [Frisch et al. 2008].

Search We will now study how a constraint programming solver will search for the solutions given the above model and how propagation will interact with



(a) Non-reified (b) Reified

Figure 3.1: Search-propagation interaction of the non-reified frequent itemset model (left) and the reified frequent itemset model (right). A search step is indicated with a circle around the variable, and a propagation step with a square.

it. A first observation is that the set of transactions is completely determined by the itemset, so we only need to search over the item variables.

When an item variable is set ($D(I_i) = \{1\}$) by the search, only the constraints that contain this item will be activated. In other words, the frequency constraint will not be activated, but every coverage constraint that contains this item will be. A coverage constraint is a *reified summation constraint*, for which the propagator was explained in Section 2.2.2. In summary, when an item variable is set, the following propagation is possible for the coverage constraint:

- if for some t: $\sum_{i \in \mathcal{I}} (1 - \mathcal{D}_{ti}) * (\min D(I_i)) > 0$ then remove 1 from $D(T_t)$;
- if for some t: $\sum_{i \in \mathcal{I}} (1 - \mathcal{D}_{ti}) * (\max D(I_i)) = 0$ then remove 0 from $D(T_t)$.

Once the domain of a variable T_t is changed, the frequency constraint is activated. The frequency constraint is a *summation constraint*, that checks whether:

$$\sum_{t \in \mathcal{S}} (\max D(T_t)) \geq \theta.$$

If this constraint fails, we do not need to branch further and we can backtrack.

Example 5. *Figure 3.1(a) shows part of a search tree for a small example with a minimum frequency threshold of 2. Essentially, the search first tries to add an item to an itemset and after backtracking it will only consider itemsets not including it. After a search step (indicated in green), the propagators are activated. The coverage propagators can set transactions to* 0 *or* 1*, while the frequency constraint can cause failure when the desired frequency can no longer be obtained (indicated by a red cross in the two left-most branches).*

Observe that in the example we branched on item 2 first. This follows the generic 'most constrained' variable order heuristic, which branches over the variable contained in most constraints first (remember that the coverage constraints are posted on items that have a 0 in the matrix). If item 1 was branched over first, the search tree would be larger, as both branches would have to determine separately that $I_2 = 1$ does not result in a frequent itemset. An experimental investigation of different branching heuristics is done in Section 3.6.

3.2.2 Improved Model

Inspired by observations in traditional itemset mining algorithms, we propose an alternative model that substantially reduces the size of the search tree by introducing fine-grained constraints. The main observation is that we can formulate the frequency constraint on each item individually:

Property 3 (Reified Frequency Constraint)**.** *Given a database $\mathcal{D}$, an itemset $I \neq \emptyset$ and a transaction set T, such that $T = \varphi_{\mathcal{D}}(I)$, then*

$$|T| \geq \theta \Longleftrightarrow \forall i \in \mathcal{I} : I_i = 1 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta. \tag{3.5}$$

where $I_i, T_t \in \{0, 1\}$ and $I_i = 1$ iff $i \in I$ and $T_t = 1$ iff $t \in T$.

Proof. We observe that we can rewrite $\varphi_{\mathcal{D}}(I)$ as follows:

$$\varphi_{\mathcal{D}}(I) = \{t \in \mathcal{S} | \forall i \in I : \mathcal{D}_{ti} = 1\} = \bigcap_{i \in I} \varphi_{\mathcal{D}}(\{i\})$$

Using this observation, it follows that:

$$\begin{aligned}
|T| \geq \theta &\Longleftrightarrow \left| \bigcap_{j \in I} \varphi_{\mathcal{D}}(\{j\}) \right| \geq \theta \\
&\Longleftrightarrow \forall i \in I : \left| \varphi_{\mathcal{D}}(\{i\}) \cap \bigcap_{j \in I} \varphi_{\mathcal{D}}(\{j\}) \right| \geq \theta \\
&\Longleftrightarrow \forall i \in I : |\varphi_{\mathcal{D}}(\{i\}) \cap T| \geq \theta \\
&\Longleftrightarrow \forall i \in \mathcal{I} : I_i = 1 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta.
\end{aligned}$$

□

The improved model consists of the coverage constraint (equation (3.2)) and the newly introduced reified frequency constraint (equation (3.5)). This model is equivalent to the original model, and also finds all frequent itemsets.

The reified frequency constraint is posted on every item separately, resulting in more fine-grained search-propagation interactions. Essentially, the reified frequency constraint performs a kind of *look-ahead*; for each item, a propagator will look ahead to determine whether adding this item to the itemset would make it infrequent. If it does, the item will be removed from further consideration as its inclusion in the set can never result in a valid solution. In summary, the main additional propagation allowed by the reified constraint is the following:

- if for some i: $\sum_{t \in \mathcal{S}} \mathcal{D}_{ti} * (\max D(T_t)) < \theta$ then remove 1 from $D(I_i)$.

Example 6. *Figure 3.1 shows the search trees of both the original non-reified model as well as the improved model using the reified frequency constraint.*

In the original model (Figure 3.1(a)), the search branches over $I_2 = 1$*, after which the propagation detects that this makes the itemset infrequent and fails (left-most branch). In the reified model (Figure 3.1(b)) the reified frequency propagator for* I_2 *detects that this item is infrequent. When evaluating the sum* $\big(0 * (\max D(T_1)) + 1 * (\max D(T_2)) + 0 * (\max D(T_3))\big)$*, it is easy to see that the maximum is* $1 < 2$*, leading to* $I_2 = 0$ *(second level). The same situation occurs for* I_3 *near the bottom of the figure. This time, the propagator takes into account that at this point* $T_3 = 0$ *and hence* $\max D(T_3) = 0$*.*

The reified propagations avoid creating branches that can only fail. In fact, using the reified model, the search becomes failure-free: every branch will lead to a solution, namely a frequent itemset. This comes at the cost of a larger number of propagations. In Section 3.6 we experimentally investigate the difference in efficiency between the two formulations.

3.2.3 Comparison

Let us now study how the proposed CP-based approach compares to traditional itemset mining algorithms. We discussed such approaches in Section 2.1.1 and provided the outline of a general depth-first frequent itemset mining algorithm (Algorithm 2.1 on page 15).

Comparison with search using the CP model We here restrict our discussion to a comparison of high level principles. A more detailed analysis, including the data structures of specific constraint programming systems, is given in Chapter 4 in the context of creating a specialised constraint programming/itemset mining system.

We first consider the differences in the search trees when using our CP model as compared to traditional mining algorithms. These differences can be understood by comparing the search tree of a depth-first search algorithm (Figure 2.3 on page 16) with the search tree of a constraint programming system (Figure 3.1 on page 42). In depth-first itemset mining, each node in the search tree corresponds to an itemset. Search proceeds by adding items to it; nodes in the search tree can have an arbitrary number of children. In CP, each node in the search tree corresponds to a domain, which in our model represents a choice for possible values of items and transactions. Search proceeds by restricting the domain of a variable. The resulting search tree is always binary, as every item

is represented by a boolean variable that can either be true or false (include or exclude the item).

We can identify the following relationship between nodes in the search tree of a CP system and nodes in the search tree of itemset miners. Denoting by $D(I_i)$ the domain of item variable I_i in the state of the CP system, we can map each state to an itemset as follows:

$$\{i \in \mathcal{I} \mid D(I_i) = \{1\}\}.$$

Essentially, in CP some branches in the search tree correspond to an assignment $D(I_i) = \{0\}$ for an item i (i.e. the item i is removed from consideration). All nodes across a path of such branches are collapsed in one node of the search tree of the itemset miner, turning the binary tree into an $n-$ary tree.

Even though it might seem that this different perception of the search tree leads to a higher memory use in CP systems, this is not necessarily the case. If the search tree is traversed in the order indicated in Figure 3.1(b), once we have assigned value $D(I_1) = \{0\}$ and generated the corresponding child node, we no longer need to store the original domain D with $D(I_1) = \{0, 1\}$. The reason is that there are no further children to generate for this original node in the search tree; if the search returns to this node, we can immediately backtrack further to its parent (if any). Hence, additional memory only needs to be consumed by branches corresponding to $D(I_i) = \{1\}$ assignments and not for $D(I_i) = \{0\}$. This implies that in practice the efficiency of the approach depends on the implementation of the CP system; it does not depend on the theoretically different shape of the search tree.

In more detail these are the possible domains for the variables representing items during the search of the CP system:

- $D(I_i) = \{0, 1\}$: this represents an item that can still be added to the itemset, but that currently is not included; in traditional itemset mining algorithms, these are the items included in the projected database;
- $D(I_i) = \{0\}$: this represents an item that will not be added to the itemset. In the case of traditional itemset mining algorithms, these are items that are neither part of the projected database nor part of the current itemset;
- $D(I_i) = \{1\}$: this represents an item that will be part of all itemsets deeper down the search tree; in the case of traditional algorithms, this item is part of the itemset represented in the search tree node.

Similarly, we have the transaction variables:

- $D(T_t) = \{0, 1\}$: this represents a transaction that is covered by the current itemset (since 1 is still part of its domain), but may still be removed from the coverage later on; in traditional algorithms, this transaction is part of the projected database;
- $D(T_t) = \{0\}$: this represents a transaction that is not covered by the current itemset; in traditional algorithms, this transaction is not part of the projected database;
- $D(T_t) = \{1\}$: this represents a transaction that is covered by the current itemset and that will be covered by all itemsets deeper down the search tree, as the transaction contains all items that can still be added to the itemset; in traditional itemset mining algorithms, this transaction is part of the projected database.

A second difference is hence which information is available about transactions during the search. In our CP formalization, we distinguish transactions with domain $D(T_t) = \{0, 1\}$ and $D(T_t) = \{1\}$. Frequent itemset mining algorithms do not make this distinction. This difference allows one to determine when transactions are *unavoidable*: A transaction becomes unavoidable ($D(T_t) = \{1\}$) if all remaining items ($1 \in D(I_i)$) are included in it; the propagation to detect this occurs in branches where items are removed from consideration. Such branches are not present in traditional itemset mining algorithms.

Thirdly, to evaluate the constraints, CP systems store constraints or propagators during the search. Essentially, to every node in the search tree a *state* is associated that reflects active constraints, propagators and variables. Such a state corresponds to the concept of a projected database in itemset mining algorithms. The data structures for storing and maintaining propagators in CP systems and in itemset mining algorithms are however often very different. Such differences are very solver specific and are outside of the scope of this chapter.

Overall, the comparison shows that there are many high-level similarities between itemset mining and constraint programming systems, but that in many cases one can also expect lower-level differences. Our experiments will show that these low-level differences can have a significant practical impact. This will motivate us to develop a hybrid system in the next chapter that bridges the gap between the two.

3.3 Closed Itemset Mining

Condensed representations aim at reducing redundancy in the solutions of an itemset mining algorithm. Closed itemsets were introduced in Section 2.1.2 and are the premier form of condensed representation. They provide a lossless compression of the total set of (frequent) itemsets: using only the closed itemsets the entire collection of frequent itemsets can be reconstructed [Pasquier et al. 1999]. See [Calders and Goethals 2007] for an overview of condensed representations and their compression.

Using Definition 5 of closed itemset mining, the problem of frequent closed itemset mining can be defined as follows.

Definition 9 (Frequent Closed Itemset Mining)**.** *Given a database $\mathcal{D}$ and a threshold θ, the frequent closed itemset mining problem consists of computing*

$$\{I \mid I \subseteq \mathcal{I}, \mathit{frequency}_{\mathcal{D}}(I) \geq \theta, \psi_{\mathcal{D}}(\varphi_{\mathcal{D}}(I)) = I\}.$$

We will now formulate this problem in CP, building on the improved model of the frequent itemset mining problem in the previous section.

3.3.1 Constraint Programming Model

Compared to frequent itemset mining, the additional constraint that we need to express is the closedness constraint. We can deal with this constraint in a similar way as with the coverage constraint. Assuming that T represents the set of transactions covered by an itemset I, the constraint that we need to check is the following:

$$\psi_{\mathcal{D}}(T) = I, \tag{3.6}$$

as in this case $\psi_{\mathcal{D}}(\varphi_{\mathcal{D}}(I)) = I$. This leads to the following constraint in the CP model, which should be posted together with the constraints in equation (3.2) and equation (3.5).

Property 4 (Closedness Constraint)**.** *Given a database $\mathcal{D}$, an itemset I and a transaction set T, such that $T = \varphi_{\mathcal{D}}(I)$, then*

$$I = \psi_{\mathcal{D}}(T) \Longleftrightarrow \forall i \in \mathcal{I} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{S}} T_t(1 - \mathcal{D}_{ti}) = 0, \tag{3.7}$$

where $I_i, T_t \in \{0, 1\}$ and $I_i = 1$ iff $i \in I$ and $T_t = 1$ iff $t \in T$.

The proof is similar to the proof for the coverage constraint.

3.3.2 Comparison

Several classes of algorithms have been proposed for closed itemset mining, each being either an extension of a breadth-first algorithm such as Apriori [Agrawal et al. 1996], or a depth-first algorithm, operating on tid-lists [Zaki et al. 1997] or FP-trees [Han et al. 2000]. As in the previous section, we limit ourselves here to depth-first mining algorithms.

Initial algorithms for mining closed itemsets were based on a *repository* in which closed itemsets were stored. The search is performed by a depth-first frequent itemset miner which is modified as follows:

- when it is found that all transactions in a projected database contain the same item, this item is immediately added to the itemset as without it the itemset cannot be closed;
- for each itemset I_1 found in this way, it is checked in the repository whether an itemset $I_2 \supseteq I_1$ has been found earlier which has the same frequency; if not, the itemset is stored in the repository and the search continues; otherwise the closed supersets, starting with I_2, have already been found earlier as children of I_2 so this branch of the search tree is pruned.

The first modification only checks items that are in the projected database, which are by construction items $i > max(I)$ that are higher in the lexicographic order. The repository is needed to check whether there is no superset with an additional item $i < max(I)$; this is what the second modification does. With an appropriate search order only closed itemsets are stored [Pei et al. 2000].

This procedure works well if the number of closed sets is small and the database is large. When the number of closed itemsets is large, storing itemsets and searching in them can be costly. The LCM algorithm addresses this problem [Uno et al. 2005]. In this algorithm also for the items $i < \max(I)$ it is checked in the data whether they should be part of the closure, even though the depth-first search procedure does not recurse on such items.

Constraint Propagation The additional constraint (3.7) for closed itemset mining is similar to the coverage constraint and hence its propagation is also similar. When all remaining transactions (i.e. those for which $1 \in D(T_t)$) contain a certain item, the propagator will:

- change the domain of the item i to $D(I_i) = \{1\}$ if $1 \in D(I_i)$;

- fail if $1 \notin D(I_i)$.

Hence, in this case we do not have failure-free search; if the closure constraint requires the inclusion of an item in the closure that cannot be included, the search will backtrack.

Overall this behaviour is very similar to that of the LCM algorithm: essentially we are performing a backtracking search without storing solutions, in which items in the closure are immediately added and some branches are pruned as they fail to satisfy an order constraint. The main difference between LCM and the CP system is as in the previous section: other data structures are used and the search tree is differently organized.

3.4 Discriminative Itemset Mining

The task in discriminative itemset mining is to find itemsets that allow one to discriminate two databases from each other. As it turns out, integrating this constraint efficiently in constraint programming requires the addition of a new primitive to the constraint programming system that we used till now. On the one hand this shows the limits of the declarative primitives presented till now; on the other hand, our results demonstrate the feasibility of adding new data mining primitives as *global constraints*. Furthermore, as we will see, the application of the CP principles in the development of a new constraint propagator turns out to be crucial in improving the performance of existing mining systems.

3.4.1 Problem Definition

We define the *stamp point* of an itemset I as

$$\sigma(I) = \big(\mathit{frequency}_{\mathcal{D}^+}(I), \mathit{frequency}_{\mathcal{D}^-}(I)\big). \tag{3.8}$$

Hence the stamp point of an itemset is a vector (p, n) where p is the frequency of this itemset in $\mathcal{D}^+$ and n is the frequency of this itemset in $\mathcal{D}^-$. A *discrimination score* can be computed by a function $f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|)$ over the contingency table defined by $p, n, |\mathcal{D}^+|$ and $|\mathcal{D}^-|$ (see Section 2.1.2). For itemsets, the stamp point $\sigma(I) = (p, n)$ is used to calculate the discrimination score $f(\sigma(I), |\mathcal{D}^+|, |\mathcal{D}^-|)$. χ^2 is a well-known measure of correlation in statistics

that can be used as discrimination score:

$$\chi^2(p,n,|\mathcal{D}^+|,|\mathcal{D}^-|) = \frac{(p - \frac{(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^+|)^2}{\frac{(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^+|} + \frac{(n - \frac{(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^-|)^2}{\frac{(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^-|} + \frac{(|\mathcal{D}^+| - p - \frac{|\mathcal{D}|-(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^+|)^2}{\frac{|\mathcal{D}|-(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^+|} + \frac{(|\mathcal{D}^-| - n - \frac{|\mathcal{D}|-(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^-|)^2}{\frac{|\mathcal{D}|-(p+n)}{|\mathcal{D}|}\cdot|\mathcal{D}^-|} \tag{3.9}$$

where it is assumed that $0/0 = 0$. An illustration of this measure is given in Figure 3.2. The domain of stamp points $[0,|\mathcal{D}^+|] \times [0,|\mathcal{D}^-|]$ is often called *PN-space*.

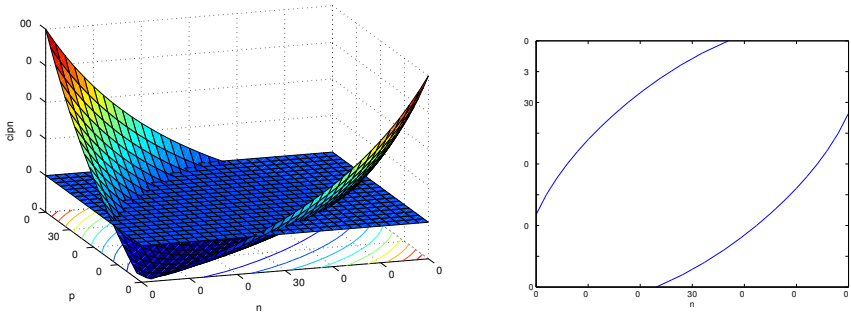


Figure 3.2: Left: Plot of the χ^2 measure in $[0,|\mathcal{D}^+|] \times [0,|\mathcal{D}^-|]$ with $|\mathcal{D}^+| = 40$ and $|\mathcal{D}^-| = 60$. The plane represents a threshold at $\chi^2 = 20$. Right: isometric of the threshold in PN space.

Essentially, we are interested in finding itemsets which are as close as possible to the maxima in one of the opposite corners; the χ^2 measure scores higher the closer we are to these maxima.

In Section 2.1.2 we introduced the problem of finding all discriminative itemsets, given a discrimination measure f and a minimum score θ. Instead of thresholding a discrimination measure, the measure can also be used as an optimisation criterion.

Definition 10 (Top-k Discriminative Itemset Mining)**.** *Given a database $\mathcal{D}$, a discrimination measure f and a value k, the top-k discriminative itemset*

mining problem is the problem of finding the first k elements of the list $[I_1, I_2, \ldots, I_n]$ consisting of all itemsets $I \subseteq \mathcal{I}$ downward sorted by their $f(\sigma(I), |\mathcal{D}^+|, |\mathcal{D}^-|)$ values.

In other words, the set of k patterns that score highest according to the discrimination measure. For $k = 1$ this corresponds to finding

$$\arg\max_{I \subseteq \mathcal{I}} f(\sigma(I), |\mathcal{D}^+|, |\mathcal{D}^-|).$$

Note that being a top-k discriminative pattern makes no statement about the statistical significance of the pattern. If a statistical score such as χ^2 is used then the top-k patterns will be the most likely patterns to pass the statistical test. However, to effectively test the significance of the patterns, one would have to apply a Bonferroni correction for multiple tests proportional to the size of the search space, or apply an additional statistical testing procedure. See [Webb 2007] for a discussion of significance testing procedures for patterns.

3.4.2 Constraint Programming Model

The discrimination constraint can be expressed in a straightforward way. In addition to the variables T_t and I_i we introduce two new variables p and n, calculated as follows:

$$p = \sum_{t \in \mathcal{S}^+} T_t \qquad n = \sum_{t \in \mathcal{S}^-} T_t, \tag{3.10}$$

where $\mathcal{S}^+$ and $\mathcal{S}^-$ represent the set of transaction identifiers in the positive database $\mathcal{D}^+$ and negative database $\mathcal{D}^-$ respectively. The discrimination constraint is now expressed as follows.

Property 5 (Discrimination Constraint)**.** *Given two databases $\mathcal{D}^+$ and $\mathcal{D}^-$, a transaction set T, a discrimination measure f and a threshold θ, an itemset is discriminative iff*

$$f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|) \geq \theta,$$

where p and n are defined as described in equation (3.10). We will abbreviate $f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|)$ to $f(p, n)$ in case $\mathcal{D}^+$ and $\mathcal{D}^-$ are clear from the context.

Such a constraint could be readily expressed in CP systems; essentially, a discrimination measure $f(p, n, |\mathcal{D}^+|, |\mathcal{D}^-|)$, such as χ^2, is composed of a large number of mathematical operations on the variables p, n, $|\mathcal{D}^+|$ and $|\mathcal{D}^-|$. By carefully decomposing the measure into simple operations using intermediate

variables, CP systems may be able to calculate a bound on the function and maintain bound consistency. However, this approach would be cumbersome (for instance, in the case of the χ^2 function we would need to rewrite the formula to take care of the division by zero) and it is not guaranteed that rewriting its formula leads to an efficient computation strategy for all discrimination measures.

Hence, we propose a more robust approach here, which requires the addition of a new constraint in a CP system to enable the maintenance of tighter bounds for discrimination measures with 'nice' properties. Adding specialized *global constraints* is common practice in CP [Rossi et al. 2006] and hence well supported in systems such as Gecode. The main observation that we use in this case is that many discrimination measures, such as χ^2, are *zero on the diagonal and convex (ZDC).*

Definition 11. *A scoring function f is* zero diagonal convex (ZDC) *if it has the following two properties:*

- *the function reaches its minimum in all stamp points on the diagonal in PN-space, i.e.,*

$$\forall 0 \leq \alpha \leq 1 : f(\alpha|\mathcal{D}^+|, \alpha|\mathcal{D}^-|) = 0.$$

- *the function is* convex, *i.e., for every pair of stamp points $\sigma \neq \sigma'$ it holds that*

$$\forall 0 \leq \alpha \leq 1 : f(\alpha\sigma + (1-\alpha)\sigma') \leq \alpha f(\sigma) + (1-\alpha) f(\sigma').$$

Theorem 1. *Fisher score, information gain, gini index, and χ^2 are ZDC measures.*

Definitions, as well as independent, alternative proofs of this theorem, can be found in [Cheng et al. 2007; Morimoto et al. 1998; Morishita and Sese 2000]. The plot of χ^2 in Figure 3.2 illustrates these two properties: the function is zero on the diagonal and convex.

For a ZDC measure the following can be proved.

Theorem 2 (Maximum for ZDC Measures)**.** *Let f be a ZDC measure and $0 \leq p_1 \leq p_2$ and $0 \leq n_1 \leq n_2$. Then*

$$\max_{(\sigma_1,\sigma_2)\in[p_1,p_2]\times[n_1,n_2]} f(\sigma_1, \sigma_2) = \max\{f(p_1, n_2), f(p_2, n_1)\}$$

Proof. The proof is similar to that of [Morishita and Sese 2000]. First, we observe that the function is *convex*. Hence, we know that the maximum in a

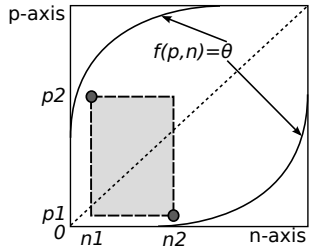


Figure 3.3: Illustration of a rectangle of stamp points in PN space; within the rectangle $[p_1, p_2] \times [n_1, n_2]$, a ZDC measure reaches its highest value in one of the two highlighted stamp points

space $[p_1, p_2] \times [n_1, n_2]$ is reached in one of the points (p_1, n_1), (p_1, n_2), (p_2, n_1) and (p_2, n_2). Next, we need to show that we can ignore the corners (p_1, n_1) and (p_2, n_2). Observing that the minimum is reached on the diagonal, we can distinguish several cases.

If $n_1/|\mathcal{D}^-| < p_1/|\mathcal{D}^+|$, the point (p_1, n_1) is 'below' the diagonal. We know for the point $(\frac{|\mathcal{D}^+|}{|\mathcal{D}^-|}n_1, n_1)$ on the diagonal that $f(\frac{|\mathcal{D}^+|}{|\mathcal{D}^-|}n_1, n_1) = 0$. Due to the convexity we know then that $f(\frac{|\mathcal{D}^+|}{|\mathcal{D}^-|}n_1, n_1) = 0 \leq f(p_1, n_1) \leq f(p_2, n_1)$.

Similarly, we can show that $f(p_1, n_1) \leq f(p_1, n_2)$ if (p_1, n_1) is above the diagonal; that $f(p_2, n_2) \leq f(p_2, n_1)$ if (p_2, n_2) is below the diagonal; and that $f(p_2, n_2) \leq f(p_1, n_2)$ if (p_2, n_2) is above the diagonal. □

The bound states that to find the highest possible score in a rectangle of points, it suffices to check two corners of the rectangle. This is illustrated in Figure 3.3, where a rectangle $[p_1, p_2] \times [n_1, n_2]$ is highlighted; the maximum on a ZDC measure is reached in one of the two corners (p_2, n_1) and (p_1, n_2). This property can be used to implement a propagator for a discrimination constraint.

Similar to the model for standard frequent itemset mining, we can improve the model by posting the discrimination constraint on each item individually, leading to the *reified discrimination constraint*:

$$\forall i \in \mathcal{I} : I_i = 1 \rightarrow f(\sum_{t \in \mathcal{S}^+} T_t \mathcal{D}_{ti}, \sum_{t \in \mathcal{S}^-} T_t \mathcal{D}_{ti}) \geq \theta \tag{3.11}$$

Our CP model of discriminative itemset mining is a combination of this constraint and the coverage constraint in equation (3.2).

Algorithm 3.2 is the conceptual propagator for the above constraint, obtained by applying Theorem 2, where T^{min} and T^{max} are shorthand for $\min D(T)$ and $\max D(T)$:

Algorithm 3.2 Conceptual propagator for $I_i = 1 \rightarrow f(\sum_{t\in\mathcal{S}^+} T_t\mathcal{D}_{ti}, \sum_{t\in\mathcal{S}^-} T_t\mathcal{D}_{ti}) \geq \theta$

```
1: if D(I_i) = {1} then
2:     post constraint f(Σ_{t∈S+} T_t D_ti, Σ_{t∈S-} T_t D_ti) ≥ θ
3: else
4:     upper = max{f(Σ_{t∈S+} T_t^max D_ti, Σ_{t∈S-} T_t^min D_ti),
5:                 f(Σ_{t∈S+} T_t^min D_ti, Σ_{t∈S-} T_t^max D_ti)}
6:     if upper < θ then
7:         D(I_i) = D(I_i) \ {1}
8:     end if
9: end if
```

To understand this propagator, consider the example in Figure 3.3, where we have marked the curve $f(p,n) = \theta$ for a particular value of θ. Due to the convexity of the function f, stamp points for which $f(p,n) \geq \theta$ can be found in the lower-right and upper-left corner. None of the stamp points in $(p,n) \in [p_1,p_2] \times [n_1,n_2]$, where $p_1 = \sum_{t\in\mathcal{S}^+} T_t^{min}\mathcal{D}_{ti}$, $p_2 = \sum_{t\in\mathcal{S}^+} T_t^{max}\mathcal{D}_{ti}$, $n_1 = \sum_{t\in\mathcal{S}^-} T_t^{min}\mathcal{D}_{ti}$ and $n_2 = \sum_{t\in\mathcal{S}^-} T_t^{max}\mathcal{D}_{ti}$, satisfies $f(p,n) \geq \theta$ in the figure; this can easily be checked by the propagator by determining that $f(p_1,n_2) < \theta$ and $f(p_2,n_1) < \theta$.

3.4.3 Comparison

Traditional discriminative itemset mining algorithms essentially proceed by upgrading frequent itemset mining algorithms such that a different anti-monotonic constraint is used during the search. This constraint is based on the derivation of an upper-bound on the discrimination measure [Morishita and Sese 2000].

Definition 12 (Upper-Bound)**.** *Given a function $f(p,n)$, function $g(p,n)$ is an upper-bound for f iff $\forall p,n$: $f(p,n) \leq g(p,n)$.*

In the case of itemsets it is said that the upper-bound is anti-monotonic, if the constraint $g(I) \geq \theta$ is anti-monotonic. The following upper-bound was presented by Morishita and Sese for ZDC measures [Morishita and Sese 2000].

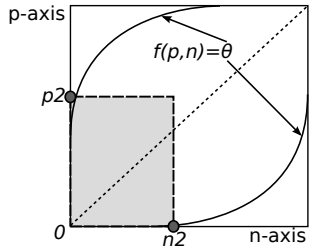


Figure 3.4: Stamp points $(p_2, 0)$ and $(0, n_2)$ are upper bounds for the itemset I with $(p_2, n_2) = \sigma(I)$.

Theorem 3 (Upper-Bound for ZDC Measures)**.** *Let $f(p, n)$ be a ZDC measure, then $g(p, n) = \max(f(p, 0), f(0, n))$ is an upper-bound for $f(p, n)$ and $g(I) \geq \theta$ is an anti-monotonic constraint.*

Proof. The fact that this function is an upper-bound follows from Theorem 2, where we take $p_1 = n_1 = 0$, $p_2 = \sigma_1(I)$ and $n_2 = \sigma_2(I)$. The anti-monotonicity follows from the fact that $f(p, 0)$ and $f(0, n)$ are monotonically increasing functions in p and n, respectively. p and n represent the frequency of the itemset in the positive and negative databases $\mathcal{D}^+$ and $\mathcal{D}^-$ respectively, which are anti-monotonic as well. □

The bound is illustrated in Figure 3.4. Given the threshold θ in this figure, the itemset I with stamp point $(p_2, n_2) = \sigma(I)$ will not be pruned, as at least one of $f(p_2, 0)$ or $f(0, n_2)$ has a discrimination score that exceeds the threshold θ.

This bound is used in an upgraded frequent itemset miner, of which the main differences compared to a standard frequent itemset miner are:

- we need to be able to compute the frequency in the two classes of data separately. This can be achieved both when using *tid-list* or *FP-trees*;
- to prune items from the projected database, instead of a frequency constraint, a constraint on the upper-bound of the discrimination score is used: a subtree of the search tree is pruned iff $g(I) < \theta$, where θ is the threshold on the score.

In case we do not wish to find all discriminative patterns with a score above θ, but instead the top-k patterns with the highest discriminative score, a branch-and-bound search strategy can be employed. In top-1 branch-and-bound search,

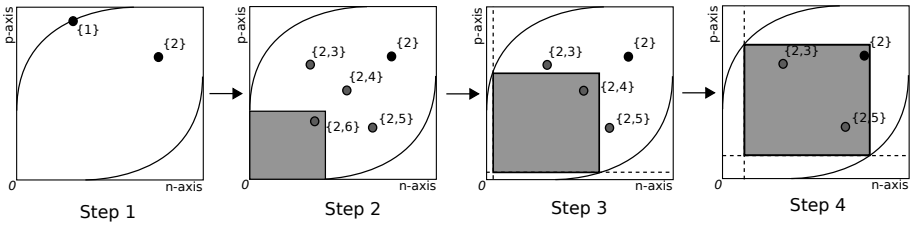


Figure 3.5: Illustration of the possible propagation for discriminative itemset mining; this propagation loop was not yet studied in specialized itemset mining algorithms.

the bound on the discrimination score $f(p, n)$ is increased as patterns with a higher score are found. For top-k branch-and-bound search, the bound is set to that of the k-th pattern.

Constraint propagation Intuitively, when we compare Figures 3.3 and 3.4, we can see that the search would continue for the itemset in Figure 3.4 because the maximum reachable score is measured in the points $(p_2, 0)$ and $(0, n_2)$, for which the score is above the threshold θ. On the other hand the search would stop in Figure 3.3 because the maximum the itemset can reach is measured in (p_2, n_1) and (p_1, n_2), for which the score is below the threshold. The difference is that in Figure 3.3, p_1 and n_1 are taken into account, which is the number of *unavoidable* transactions. As outlined on page 47, unavoidable transactions are transactions for which $\min D(T_t) = 1$. So instead of having to use the upper-bound of Theorem 3, which does not take unavoidable transactions into account, we can use Theorem 2, which offers a much tighter bound, especially in the case of many unavoidable transactions.

Using the reified discrimination constraint leads to fine-grained interaction between search and propagation similar to the reified frequency constraint in Section 3.2.2; excluding an item from the itemset by reducing its domain to $D(I_i) = \{0\}$, can lead to the following propagation loop:

1. some transactions become unavoidable and are changed to $D(T_t) = \{1\}$;
2. $D(T_t)$ having changed, the reified discrimination constraints are checked; possibly a constraint detects that some item can no longer be included in the itemset and the item's domain is reduced to $D(I_i) = \{0\}$;
3. setting a $D(I_i)$ to 0 can lead to unavoidable transactions, return to step 1.

Example 7. *Figure 3.5 illustrates this propagation loop. Assume that itemset* $\{1\}$ *is the best scoring itemset found so far, Step 1 shows this itemset's stamp point and the isometric of the scoring function in PN space. The itemset which we are refining is itemset* $\{2\}$*, shown in step 1 as well. The possible items which we can add to this itemset are* 3*,* 4*,* 5 *and* 6*. The propagation of the reified discrimination constraints (Equation 3.11) verifies for each item whether adding it to the itemset can lead to a new best itemset. Assuming there are no unavoidable transactions at this point, the dark grey box in step 2 illustrates the area that cannot lead to a new best itemset; if an itemset's stamp point is in this area, so will the stamp points of all its supersets (adding extra items will only decrease the positive/negative examples covered). The stamp point of itemset* $\{2, 6\}$ *falls in this area, hence the reified discrimination propagator removes item* 6 *from consideration. Imagine that by removing this item, some positive and negative transaction become unavoidable. Step 3 shows in dark grey the new area in which an itemset cannot lead to a higher scoring itemset. As the stamp point of* $\{2, 4\}$ *falls in this area, item* 4 *is removed from consideration as well. Imagine now that this again leads to more unavoidable transactions. As a result, the prunable region is increased, and we find that no item can be added and improve the discrimination score; we prune the search for all itemsets below* $\{2\}$*.*

This type of propagation is absent in traditional discriminative itemset miners, which use the more simple upper bound $g(I)$. We will experimentally verify whether it is beneficial to perform the additional proposed propagation in Section 3.6.

3.5 Itemset Mining with Costs

Several papers have studied alternative constraints to the frequency constraint, which has lead to the concepts of monotonic, anti-monotonic and convertible anti-monotonic constraints. The prime example on which these concepts have been illustrated both in theory and in practice are constraints in which a cost, or weight, is associated with every item. In this section, we review these constraints and then show how they can be handled in the constraint programming approach.

3.5.1 Problem Definition

Essentially, every item i now has an associated weight $c(i)$, often called the *cost*[2] of the item. Let us now define the total cost of an itemset as

$$c(I) = \sum_{i \in I} c(i).$$

Then we may be interested in finding itemsets for which we have a high total cost [Pei et al. 2001; Bucila et al. 2003; Bonchi and Lucchese 2007].

Definition 13 (Frequent Itemset Mining with Minimum Total Cost)**.** *Given a database $\mathcal{D}$ and two parameters θ and γ, the frequent itemset mining problem under a minimum cost constraint is the problem of finding the itemsets in*

$$\{I \mid I \subseteq \mathcal{I}, frequency_{\mathcal{D}}(I) \geq \theta, c(I) \geq \gamma\}.$$

Similarly, we can mine under maximum cost constraints and average cost constraints.

Definition 14 (Frequent Itemset Mining with Maximum Total Cost)**.** *Given a database $\mathcal{D}$ and two parameters θ and γ, the frequent itemset mining problem under a maximum cost constraint is the problem of finding the itemsets in*

$$\{I \mid I \subseteq \mathcal{I}, frequency_{\mathcal{D}}(I) \geq \theta, c(I) \leq \gamma\}.$$

Definition 15 (Frequent Itemset Mining with Minimum Average Cost)**.** *Given a database $\mathcal{D}$ and two parameters θ and γ, the frequent itemset mining problem under a minimum average cost constraint is the problem of finding the itemsets in*

$$\{I \mid I \subseteq \mathcal{I}, frequency_{\mathcal{D}}(I) \geq \theta, c(I)/|I| \geq \gamma\}.$$

Please note that a special case of cost-based itemset mining is achieved when $c(i) = 1$ for all i. These constraints are usually referred to as *size* constraints.

3.5.2 Constraint Programming Model

In the constraint programming framework we do not distinguish constraints by being monotonic or anti-monotonic with respect to set inclusion. Rather, constraints calculate bounds on the domains of variables, which is not tied to

[2]This terminology is again from the supermarket setting, where the cost of an item could be its price or profit.

set inclusion/exclusion specifically. Note that from a constraint programming perspective, every constraint is monotonic with respect to the domain of the variables, as a propagator can only remove values from it.

In analogy to the frequency constraint, cost constraints can be expressed in two ways, non-reified and reified. Cost constraints can be added to the usual frequency and coverage constraints in the CP model.

Property 6 (Non-reified Minimum and Maximum Total Cost Constraint)**.** *Given a database $\mathcal{D}$, an itemset I and a threshold γ, then*

$$c(I) \lesseqgtr \gamma \Longleftrightarrow (\sum_{i \in \mathcal{I}} I_i c(i) \lesseqgtr \gamma), \tag{3.12}$$

where $\lesseqgtr \in \{<, \leq, \geq, >\}$, $I_i \in \{0, 1\}$ and $I_i = 1$ iff $i \in I$.

Property 7 (Reified Minimum and Maximum Total Cost Constraint)**.** *Given a database $\mathcal{D}$, an itemset I and a threshold γ, if $frequency_{\mathcal{D}}(I) \geq 1$ then*

$$c(I) \lesseqgtr \gamma \Longleftrightarrow (\forall t \in \mathcal{S} : T_t = 1 \rightarrow \sum_{i \in \mathcal{I}} I_i D_{ti} c(i) \lesseqgtr \gamma), \tag{3.13}$$

where $\lesseqgtr \in \{<, \leq, \geq, >\}$, $I_i \in \{0, 1\}$, $I_i = 1$ iff $i \in I$ and $T_t = 1$ iff $t \in T$.

Proof. This follows from the assumption that also the coverage constraint should hold; hence if $T_t = 1$ we know that for all i with $I_i = 1$ we must have $D_{ti} = 1$. Because $frequency_{\mathcal{D}}(I) \geq 1$, we know that there is at least one transaction for which $T_t = 1$. □

Average cost constraints can be expressed by allowing for negative coefficients.

Property 8 (Non-reified Minimum and Maximum Average Cost Constraint)**.** *Given a database $\mathcal{D}$, an itemset I and a transaction set T, then*

$$c(I)/|I| \lesseqgtr \gamma \Longleftrightarrow (\sum_{i \in \mathcal{I}} I_i (c(i) - \gamma) \lesseqgtr 0), \tag{3.14}$$

where $\lesseqgtr \in \{<, \leq, =, \neq, \geq, >\}$, $I_i \in \{0, 1\}$ and $I_i = 1$ iff $i \in I$.

Proof. This follows from the following observation:

$$c(I)/|I| \lesseqgtr \gamma \Leftrightarrow c(I) \lesseqgtr \gamma |I| \Leftrightarrow (c(I) - \gamma |I|) \lesseqgtr 0.$$

□

The reified average cost constraints are obtained in a similar way as the reified total cost constraints.

3.5.3 Comparison

In the anti-monotonicity framework, maximum total cost is known to be an anti-monotonic constraint, while minimum total cost is a monotonic constraint. Average cost on the other hand is neither anti-monotone or monotone, but convertible anti-monotone. We now discuss how traditional mining algorithms deal with minimum total cost and average total cost, while pointing out the relation to our models in CP.

Minimum Total Cost Constraint: Simple Approach The *simplest* depth-first algorithms developed in the data mining community for dealing with monotonic constraints are based on the observation that supersets of itemsets satisfying the constraint also satisfy the constraint. Hence, during the depth-first search procedure, we do not need to check the monotonic constraint for children of itemsets satisfying the monotonic constraint [Pei and Han 2000]. To emulate this behaviour in CP, we would only check the satisfiability of the monotone constraint, and refrain from possibly propagating over variables. This would result in branches of the search tree being cut when they can no longer satisfy the constraint; the constraint would be disabled once it can no longer be violated.

Minimum Total Cost Constraint: DualMiner/Non-reified More advanced is the specialized *DualMiner* algorithm [Bucila et al. 2003]. DualMiner associates a triplet $(I_{in}, I_{check}, I_{out})$ with every node in the depth-first search tree of an itemset miner. Element I_{in} represents the itemset to which the node in the search tree corresponds; I_{check} and I_{out} provide additional information about the search node. Items in I_{check} are currently not included in the itemset, but may be added in itemsets deeper down the search tree; these items are part of the projected database. For items in I_{out} it is clear that they can no longer be added to any itemset deeper down the search tree. Adding an item of I_{check} to I_{in} leads to a branch in the search tree. An iterative procedure is applied to determine a final triplet $(I_{in}, I_{check}, I_{out})$ for the new search node, and to determine whether the recursion should continue:

- it is checked whether the set I_{in} satisfies all anti-monotonic constraints. If not, stop.
- it is checked which individual items in I_{check} can be added to I_{in} and satisfy the anti-monotonic constraints. Only those that do satisfy the constraints are kept in I_{check}, others are moved to I_{out}.

- it is checked whether the set $I_{in} \cup I_{check}$ satisfies the monotonic constraints. If not, stop. Every item $i \in I_{check}$ for which itemset $(I_{in} \cup I_{check}) \backslash \{i\}$ does not satisfy the monotonic constraints, is added to I_{in}. (For instance, if the total cost is too low without a certain item, we have to include this item in the itemset.) Finally, the procedure is iterated again to determine whether I_{in} still satisfies the anti-monotonic constraints.

If the loop reaches a fixed point and items are still left in I_{check} the search continues, unless it also appears that $I_{in} \cup I_{check}$ satisfies the anti-monotonic constraints and I_{in} satisfies the monotonic constraints; in this case the sets $I_{check} \subseteq I \subseteq I_{in} \cup I_{check}$ could immediately be listed.

A similar search procedure is obtained when the cost constraints are formulated in a non-reified way in the CP system. As pointed out earlier, a non-reified minimum or maximum cost constraint takes the following form:

$$\sum_{i \in \mathcal{I}} I_i c(i) \lesseqgtr \gamma.$$

Propagation for this constraint is of the following kind:

- if according to current domains the constraint can only be satisfied when the CP system includes (minimum cost constraint) or excludes (maximum cost constraint) an item, then the system does so; this corresponds to moving an item to the I_{in} or I_{out} set in DualMiner;
- if according to current domains the constraint can no longer be satisfied, backtrack;
- if according to current domains the constraint will always be satisfied, the constraint is removed from the constraint store.

Hence, the overall search strategy for the non-reified constraint is similar to that of DualMiner. There are also some differences. DualMiner does not aim at finding the transaction set for every itemset. If it finds that I_{in} satisfies the monotonic constraint and $I_{in} \cup I_{check}$ satisfies the anti-monotonic constraint, it does not continue searching, and outputs the corresponding range of itemsets explicitly or implicitly. The CP system will continue enumerating all itemsets in the range in order to find the corresponding transaction sets explicitly.

Minimum Total Cost Constraint: FP-Bonsai/Reified The main observation on which the FP-Bonsai algorithm [Bonchi and Goethals 2004] algorithm is based is that a transaction which does not satisfy the minimum cost constraints,

will never contain an itemset that satisfies the minimum cost constraint. Hence, such transactions can be removed from consideration. This may reduce the frequency of items in the projected database and result in the removal of more items from the database. The reduction in size of some transactions may trigger a new step of propagation. This continues until a fixed point is reached.

If we consider the reified minimum cost constraint,

$$T_t = 1 \rightarrow \sum_i I_i c(I) \mathcal{D}_{ti} \geq \gamma,$$

we can observe a similar propagation. Propagation essentially removes a transaction from consideration when the constraint can no longer be satisfied on the transaction. The removal of this transaction may affect the frequency of some items, requiring the propagators for the frequency constraints to be re-evaluated.

Note that the reified constraint is less useful for a **maximum total cost constraint**, $c(I) \leq \gamma$. Essentially, for every transaction only items already included in the itemset should be considered. If the sum of these items is too large, the transaction is removed from consideration. This would happen for all transactions separately, leading to a failed branch. Overall, the propagation is expensive to evaluate (as it needs to be done for each transaction) and not as effective as the non-reified propagator which can also prune items from consideration.

Minimum and Maximum Average Cost Constraints Average cost constraints convertible (anti-)monotonic (see Definition 4 on page 18 of the background chapter). They are neither monotonic nor anti-monotonic but can be converted to it by considering the items in a fixed order.

Taking average cost as an example, if the items are ordered according to increasing cost $c(I)$, when adding items that are more expensive than the current items, the average cost can only increase. For a decreasing order in item cost, the minimum average cost constraint is hence convertible anti-monotonic. Different orderings will not result in anti-monotonic behaviour, i.e. if after adding expensive items an item with a low cost would be added, the average cost would go down. Note that a conjunction of maximum and minimum cost constraints is hence not convertible anti-monotonic, as we would need opposing orders for each of the two constraints.

Essentially our formalization in CP of average cost constraints is similar to that of total cost constraints, the main difference being that negative costs are allowed. Consequently, depending on the constraint (minimum or maximum)

and the weight (positive or negative) either the maximum value in the domain or the minimum value in the domain is used in the propagator. In the non-reified form, we obtain propagation towards the items; in the reified form towards the transactions.

This search strategy is fundamentally different from the search strategy used in specialized mining systems, in which the property is used that one average cost constraint is convertible anti-monotonic. Whereas in specialized systems the combination of multiple convertible constraints poses problems, in the CP-based approach this combination is straightforward.

Conclusions Interestingly, when comparing models in CP and algorithms proposed in the data mining community, we can see that there are many similarities between these approaches. Differences in itemset mining algorithms correspond in the CP models to using the reified or non-reified constraints. The main advantage of the constraint programming approach is that the different constraints can additionally be combined. This advantage is evident when combining convertible constraints; specialized algorithms can only optimize on one such constraint at the same time.

3.6 Experiments

In previous sections we concentrated on the conceptual differences between traditional algorithms for itemset mining and constraint programming. In the present section, we first consider several choices that have to be made when implementing itemset mining in a constraint programming framework and evaluate their influence on the performance of the mining process. More specifically, we answer the following two questions about such choices:

QA What is the difference in runtime performance between using reified versus non-reified constraints of itemset mining?

QB What is the effect of different variable orderings on the performance of itemset mining?

Further (more technical and system dependent) choices made in our implementation are explained in an appendix section at the end of this chapter, for completeness and reproducibility.

We then use the best approach resulting from the above experiments to experimentally compare our constraint programming framework CP4IM to

Dataset	# transactions	# items	density	10% freq.	# sols.
soybean	630	59	0.25	63	12754
splice-1	3190	290	0.21	319	1963
anneal	812	41	0.43	81	1891712
mushroom	8124	119	0.19	812	574514
letter	20000	74	0.33	2000	9659119

Table 3.1: Properties of the datasets used, *# sols.* is the number of solutions (frequent itemsets) at a 10% frequency threshold.

state-of-the-art itemset mining systems. More specifically, our comparative experiments focus on answering the following questions:

Q1 What is the difference in performance of CP4IM for *frequent* itemset mining and traditional algorithms?

Q2 What is the difference in performance of CP4IM for *closed* itemset mining and traditional algorithms?

Q3 Is the additional propagation in CP4IM for *discriminative* itemset mining beneficial? If so, how much?

Q4 Is the alternative approach for dealing with *convertible* constraints in CP4IM beneficial? If so, how much?

We ran experiments on PCs with Intel Core 2 Duo E6600 processors and 4GB of RAM, running Ubuntu Linux. The experiments are conducted using the Gecode constraint programming system, discussed in Section 2.2.5.

Data Sets In our experiments we used data from the UCI Machine Learning repository[3]. To deal with missing values we preprocessed each dataset in the same way as Frank and Witten [1998]: we first eliminated all attributes having more than 10% of missing values and then removed all examples (transactions) for which the remaining attributes still had missing values. Numerical attributes were binarized by using unsupervised discretization with 4 equal-frequency bins; each item for an attribute corresponds to a threshold between two bins. These preprocessed datasets can be downloaded from our website[4]. The datasets and their basic properties can be found in Table 3.1. The density is the proportion of ones in the matrix. Typically, datasets with a high density contain more item combinations than datasets with a low density.

[3] http://archive.ics.uci.edu/ml/
[4] http://dtai.cs.kuleuven.be/CP4IM/

	CP4IM	LCM	Eclat	Patternist	DDPmine
Frequent itemsets	X	X*	X	X	
Closed itemsets	X	X	X*		
Correlated itemsets	X				X
Monotone constraints	X			X	
Convertible constraints	X			X	
Combinations of the above	X			X**	

Table 3.2: Comparison of the mining tasks that different miners support. X*: Not originally designed for this task. X**: The system supports combining constraints, except multiple convertible constraints.

Hence, for itemset mining problems, datasets with a higher density can be more difficult to mine.

Alternative itemset miners We used the following state-of-the-art specialized algorithms, for which implementations are freely available, as the basis for our comparative evaluation:

LCM [Uno et al. 2005] is a specialized frequent closed itemset mining algorithm based on tid-lists;

Eclat [Zaki and Gouda 2003] is a specialized depth-first frequent itemset mining based on tid-lists;

Patternist [Bonchi and Lucchese 2007] is a specialized breadth-first algorithm for mining under monotonic and anti-monotonic constraints;

DDPMine [Cheng et al. 2007] is a specialized depth-first closed discriminative itemset mining algorithm based on FP-trees and a repository of closed itemsets; it uses a less tight bound than the bound summarized in Section 3.4.

Note that in our experiments we are not using all algorithms discussed in previous sections. The reason is that we preferred to use algorithms for which comparable implementations by the original authors were freely available (i.e., executable on the same Linux machine).

Table 3.2 provides an overview of the different tasks that these data mining systems support. The LCM and Eclat algorithms have been upgraded by their original authors to support respectively frequent itemset mining and closed itemset mining too. Patternist is a constraint-based mining algorithm which

has been carefully designed to make maximal use of monotone and convertible constraints during the search. Our CP4IM system is the only system that supports all of these constraints as well as combinations of these constraints. Furthermore, thanks to the use of a declarative constraint programming system it can easily be extended with further types of constraints. This is what we regard as the major advantage of the constraint programming methodology. It is also interesting to contrast this approach with that taken by the alternative, more procedural systems. They were typically designed to cope with a single constraint family and were later upgraded to deal with other ones too. This upgrade usually involves changing the algorithm extensively and hard-coding the new constraint in it. In contrast, in CP one might need to add a new propagator (as we have done for the discrimination constraint), but the corresponding constraint can freely be used and combined with any other current and future constraint in the system. This is essentially the difference between a declarative and a procedural approach.

On the other hand, generality and flexibility also may have a price in terms of performance. Therefore, we do not expect CP4IM to perform well on each task, but we would hope its performance is competitive when averaging over a number of tasks.

3.6.1 Non-Reified vs Reified

In Section 3.2.2 we argued that using reified frequency constraints for the standard frequent itemset mining problem can lead to more propagation. Table 3.3 shows a comparison between running the CP model with non-reified and reified frequency constraints. Two datasets were used, each with three different frequency thresholds. For the reasonably small *anneal* dataset, it can be noted that the non-reified version needs a bit less memory and propagation, but at the cost of some failed branches in the search tree. This leads to a small increase in run time. For the bigger *mushroom* dataset however, the difference is larger. For higher minimum frequency thresholds the reified pruning becomes stronger while for the non-reified formulation the cost of a failed branch increases, leading to a larger difference in runtime.

In general we have observed that using the reified frequency constraints often leads to better performance, especially for larger datasets.

		Non-reified frequency				Reified frequency			
Dataset	freq.	mem.	props.	failures	time (s)	mem.	props.	time (s)	faster
anneal	5%	2694	235462	170	46.3	2950	236382	44.7	1.04
anneal	10%	2438	221054	248	19.3	2501	224555	18.9	1.02
anneal	15%	2309	200442	298	8.4	2373	203759	8.3	1.01
mushroom	5%	17862	7737116	10383	269.5	20486	5980478	239.8	1.12
mushroom	10%	17862	4184940	3376	74.2	20229	2853248	43.4	1.71
mushroom	15%	17862	2680479	1909	40.8	19973	1589289	10.5	3.89

Table 3.3: Comparison of the reified versus non-reified formulation of the frequency constraint on 2 datasets for different frequencies. 'freq' = frequency threshold, 'mem.' = peak memory in kilobytes, 'props.' = number of propagations, 'failures' = failed leaves in the search tree. The reified formulation never has failed branches.

	arbitrary			**minimum degree**			**maximum degree**		
	mem.	props.	time	mem.	props.	time	mem.	props.	time
soybean	1860	799791	0.5	1540	3861055	3.1	1860	137666	0.2
splic-1	124431	3188139	49	126927	3772591	60	122511	2602069	41
anneal	2116	121495848	73	2116	472448674	374	1796	577719	18
mushroom	31236	344377153	365	30148	997905725	1504	26244	6232932	48

Table 3.4: Comparison of different variable ordering heuristics in terms of peak memory in kilobytes (mem.), number of propagations (props.) and time in seconds using.

3.6.2 Variable ordering

In constraint programming it is known that the order in which the variables are searched over can have a large impact on the size of the search tree, and hence the efficiency of the search. This has received a lot less attention in the itemset mining community, except in algorithms like FP-growth where a good ordering is needed to keep the FP-tree size down.

We consider three possible variable ordering strategies, for the standard frequent itemset mining problem:

arbitrary: the input order of variables is used;

minimum degree: the variable occurring in the smallest number of propagators;

maximum degree: the variable occurring in the largest number of propagators.

The comparison of the three variable orderings can be found in Table 3.4. The experiments show that choosing the variable with maximum degree leads to a large reduction in the number of propagations and runtime. The maximum degree heuristic corresponds to choosing the item with the lowest frequency, as this item occurs in the coverage constraint (equation (3.2)) of most transactions. In other words, the most efficient variable ordering strategy is a fail-first strategy that explores the most unlikely branches of the search tree first. Such a conclusion is not surprising in the constraint programming community.

3.6.3 Frequent Itemset Mining

A comparison of specialized frequent itemset miners and CP4IM is provided in Figure 3.6 for a representative number of datasets. In this figure we show run times for different frequency thresholds as it was previously found that the differences between systems can highly depend on this constraint [Goethals and Zaki 2004].

The run time of all systems is correlated to the number of patterns found (the red line). Our CP4IM model implemented in Gecode is significantly slower than the other depth-first miners, but shows similar behaviour. This indicates that indeed its search strategy is similar, but the use of alternative data structures and other choices in the constraint programming system introduces a great deal of overhead for standard frequent itemset mining.

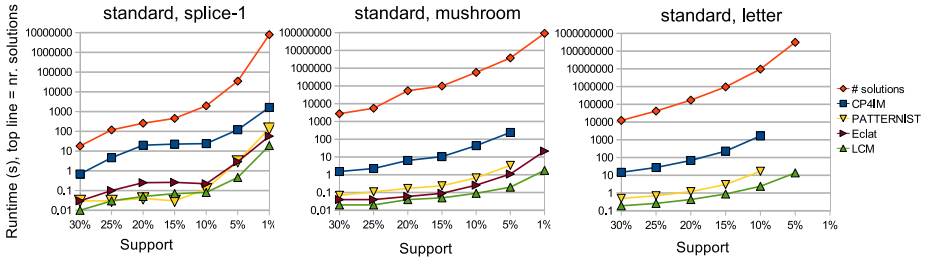


Figure 3.6: Standard itemset mining on different datasets.

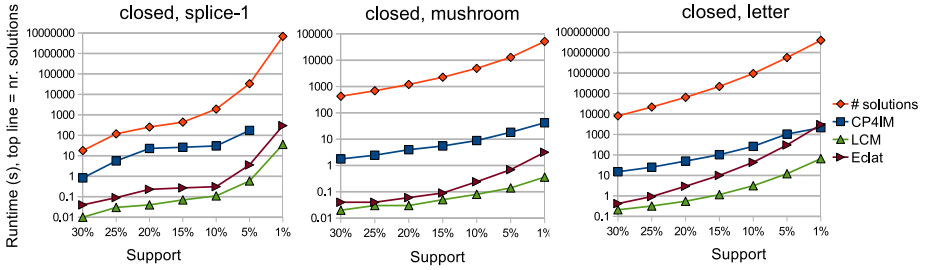


Figure 3.7: Closed itemset mining on different datasets.

3.6.4 Closed Itemset Mining

In Figure 3.7 the runtime of all mining algorithms is shown for the problem of closed itemset mining. Again, run time is correlated with the number of patterns found. We argued in the previous section that the CP system behaves similar to the LCM system. Our experiments on both the mushroom and letter data set show again that this is the case, in one case even outperforming the Eclat system, which was not originally developed for closed itemset mining.

It should be noted that on sparse data, such as mushroom, the difference in performance between Gecode and specialized systems is larger than on dense data, such as the letter data. This can be explained by the inefficient representation of sparse data in Gecode (see next chapter).

Name	Dense.	Trans.	Items	DDPmine	CP4IM	corrmine
anneal	812	93	0.45	22.46	0.22	0.02
australian-credit	653	125	0.41	3.40	0.30	0.01
breast-wisconsin	683	120	0.5	96.75	0.28	0.03
diabetes	768	112	0.5	>	2.45	0.36
german-credit	1000	112	0.34	>	2.39	0.07
heart-cleveland	296	95	0.47	9.49	0.19	0.03
hypothyroid	3247	88	0.49	>	0.71	0.02
ionosphere	351	445	0.5	>	1.44	0.24
kr-vs-kp	3196	73	0.49	125.60	0.92	0.02
letter	20000	224	0.5	>	52.66	0.65
mushroom	8124	119	0.18	0.09	14.11	0.03
pendigits	7494	216	0.5	>	3.68	0.18
primary-tumor	336	31	0.48	0.26	0.03	0.01
segment	2310	235	0.5	>	1.45	0.06
soybean	630	50	0.32	0.05	0.05	0.01
splice-1	3190	287	0.21	1.86	30.41	0.05
vehicle	846	252	0.5	>	0.85	0.07
yeast	1484	89	0.49	>	5.67	0.80

Table 3.5: Statistics of UCI datasets, and runtimes, in seconds, of two CP models and the DDPmine [Cheng et al. 2008] algorithm. > indicates that no solution was found within the timeout of 900 seconds.

3.6.5 Discriminative Closed Itemset Mining

In this experiment we compare several approaches for finding the most discriminative itemset, given labeled data. Results are shown in Table 3.5. In this setting we do not have to determine a threshold parameter. We perform experiments on a larger number of datasets; the missing values of the datasets were preprocessed in the same way as in previous experiments. However, the numerical attributes were binarized using unsupervised discretisation with 7 binary split points (8 equal-frequency bins). The resulting patterns will of the form "$X <= 5$ && $Y > 10$" instead of "$0 <= X <= 5$ && $10 <= Y <= 20$". It enforces a language bias on the patterns that is closer to that of other techniques for labeled data, such as rule learning [Fürnkranz and Flach 2005] and subgroup discovery [Grosskreutz et al. 2008]. In case of a non-binary class label, the largest class was labeled positive and the others negative. The properties of the datasets are summarized in Table 3.5; note the higher density of the datasets than in the previous experiments, resulting from the discretisation procedure.

We compare the specialised DDPmine system [Cheng et al. 2008] with the

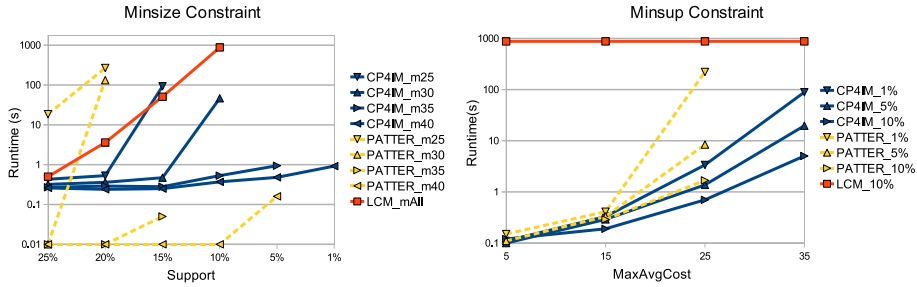


Figure 3.8: Runtimes of itemset miners on Segment data under constraints. In the left figure, the suffix _m25, _m30, etc. of the name indicates the minimum size threshold. In the right figure, the suffix _1%, etc. represent the minimum frequency threshold.

Gecode-based CP4IM system that uses the improved propagation introduced in Section 3.4.2, as well as with corrmine, our implementation of a simple Eclat-like miner that uses the same improved propagation principles as CP4IM.

Although DDPmine is a highly optimised miner, CP4IM outperforms it in most cases. This can be attributed to the more effective propagation of CP4IM. The corrmine system consistently outperforms both CP4IM and DDPmine. This shows that on the mushroom, soybean and splice-1 datasets, the better performance of DDPmine compared to CP4IM is solely due to the overhead of using a CP system; implementing the improved propagation for discriminative constraints in an itemset mining algorithm removes the overhead while maintaining the pruning gains.

3.6.6 Cost-based Itemset Mining

In this experiment we determine how CP4IM compares with other systems when additional cost constraints are employed. Results for two settings are given in Figure 3.8. In the first experiment we employed a (monotonic) minimum size constraint in addition to a minimum frequency constraint; in the second a (convertible) maximum average cost constraint. The results are positive: even though for small minimum size constraints the brute force mining algorithms, such as LCM, outperform CP4IM, CP4IM does search very effectively when this constraint selects a small number of very large itemsets (30 items or more); in extreme cases CP4IM finishes within seconds while other algorithms do not finish within our cut-off time of 30 minutes. Patternist, being

a breadth-first algorithm, was unable to finish some of these experiments due to memory problems. This indicates that CP4IM is a competitive system when the constraints require the discovery of a small number of very large itemsets. The results for convertible constraints are particularly promising, as we did not optimize the item order in any of our experiments, as is usually done when dealing with convertible constraints.

3.7 Conclusions

We started this chapter by raising the question whether constraint programming can be used for solving itemset mining problems in a declarative way. Our results show that the answer to this question is indeed positive and that the use of constraint programming offers several advantages as well as new insights. Perhaps the more important advantage is that constraint programming systems are general purpose systems supporting many different types of constraints. In this regard, we showed that it is possible to incorporate many well-known constraints, such as cost constraints, closedness or discriminative measures as defined above, as well as their combinations in the constraint programming system. The advantage of the resulting declarative approach to data mining is that it is easy to extend or change in order to accommodate new constraints, and that all constraints can automatically be combined with one another.

Furthermore, a detailed analysis of the solution strategy of constraint programming systems showed that there are many similarities between these systems and specialized itemset mining systems. Therefore, the constraint programming system arguably generalizes these systems, not only from a theoretical perspective but also from a practical one. This was confirmed in our experiments: for problems such as frequent and closed itemset mining, for which fast implementation contests were organized, these specialized systems usually outperform CP4IM; however the runtime behavior of our constraint programming approach is similar to that of the specialized systems.

The potential of the CP approach from a performance perspective was demonstrated on the problem of discriminative itemset mining. We showed that by rigorously using the principles of constraint programming, more effective propagation is obtained than in alternative state-of-the-art data mining algorithms. This confirms that it is also useful in an itemset mining context to take propagation as a guiding principle. In this regard, it might be interesting to investigate the use of alternative search strategies that have been developed in the constraint programming community [Van Hentenryck et al. 2000; Perron 1999].

Appendix: Improved Solving continued

In Section 3.6 we empirically studied the effect of non-reified versus reified formulations and of different variable ordering heuristics. In this appendix we include some additional findings, as we have experienced that making the right low-level decisions is necessary to be competitive with the highly optimized itemset mining implementations. We start by studying the differences between using boolean variables and integers with a domain of $\{0, 1\}$. We continue by studying two implementation alternatives for an essential constraint shared by all models: the coverage constraint. We end with a comparison of different value ordering heuristics; to explain and improve the results we have to provide some additional details about the Gecode system.

Apart from Gecode specific remarks, which are applicable only to solvers that do copying and cloning, the ideas and constraint decompositions presented in this appendix are also valid and applicable to other solvers. In fact, parts of the work studied here are now by default included in the aforementioned Gecode system.

Booleans vs Integers Finite domain integer solvers can choose to represent a boolean as an integer with a domain of $\{0, 1\}$, or to implement a specific boolean variable.

An essential constraint in our models is the reified summation constraint. Such a sum can be expressed both on boolean and integer variables, but the reification variable always has a boolean domain. Using boolean variables should be equally or more efficient than integer variables, especially since in our model, the integer variables need to be 'channelled' to boolean variables for use in the reification part. However, in our experiments (Table 3.6) the model using booleans was slower than the one using integers. The reason is that a given reified summation constraint on booleans B_i and boolean variable C,

$$\sum_i B_i \geq v \leftrightarrow C$$

was decomposed into two constraints: $S = \sum_i B_i$ and $S \geq v \leftrightarrow C$, where S is an additional integer variable; separate propagators were used for both constraints. For integers on the other hand, a single propagator was available. Our experiments show that decomposing a reified sum constraint over booleans into a sum of booleans and reifying the integer variable is not beneficial.

We implemented a dedicated propagator for a reified sum of boolean variables constraint, which includes an optimization inspired by SAT solvers [Gent et al.

	original boolean			integers			
Dataset	mem.	props.	time	mem.	props.	time	gain
soybean	2820	5909592	1.4	2436	1839932	0.8	1.7
splice-1	147280	23708807	129	142032	9072323	57	2.3
anneal	3140	1121904924	279	2564	273248618	136	2.1
mushroom	45636	2989128466	1447	39940	862480847	508	2.9

Table 3.6: Comparison of using boolean variables and their respective constraints versus using integer variables and constraints. 'mem.' = peak memory in kilobytes, 'props.' = number of propagations, 'time' = run time in seconds.

	integers			dedicated boolean			
Dataset	mem.	props.	time	mem.	props.	time	gain
soybean	2436	1839932	0.8	1796	1058238	0.5	1.6
splice-1	142032	9072323	57	123279	6098820	68	0.8
anneal	2564	273248618	136	2500	121495848	74	1.8
mushroom	39940	862480847	508	30852	520928196	387	1.3

Table 3.7: Comparison in propagations, peak memory and time (in seconds) of the channelled integer formulation of the base model and the boolean formulation with the dedicated propagator. 'mem.' = peak memory in kilobytes, 'props.' = number of propagations, 'time' = run time in seconds.

2006a]. A propagator is said to *watch* the variables on which it depends. A propagator is activated when the domain of one of its watched variables changes. To improve efficiency, the number of watched variables should not be larger than necessary. Assume we have a sum $\sum_{i=1}^{n} B_i \geq v \leftrightarrow C$, where all B_i and C are boolean variables, then it is sufficient to watch $\max(v, n-v+1)$ (arbitrary) variables B_i not fixed yet: the propagator can not succeed (v variables true) or fail ($n-v+1$ variables false) without assigning at least one of the watched variables. In Table 3.7 we compare the formulation of the basic frequent itemset mining problem using integers and channelling, to using boolean variables and the new dedicated propagator. As can be expected, the peak amount of memory needed when using only booleans is lower. The amount of propagations is also decreased significantly, leading to lower runtimes for all but one dataset. Hence it is overall recommended to use boolean variables with dedicated propagators.

Coverage Constraint: Propagators versus Advisers When a variable's domain changes, the propagators that watch this variable can be called with

	minimum value			maximum value		
Dataset	mem.	props.	time	mem.	props.	time
soybean	1860	137666	0.2	899	217802	0.3
splice-1	122511	2602069	41	16328	4961137	109
anneal	1796	577719	18	1412	726308	18
mushroom	26244	6232932	48	20229	9989882	63

Table 3.8: Comparison of peak memory, propagations and time (in seconds) using the minimum or maximum value ordering heuristics on the frequent itemset mining problem. 'mem.' = peak memory in kilobytes, 'props.' = number of propagations, 'time' = run time in seconds.

different amounts of information. To adopt the terminology of the Gecode system, we differentiate between classic 'propagators' and 'advisers':

- propagators: when the domain of at least one variable changes, the entire propagator is activated and re-evaluated;
- advisers: when the domain of a variable changes, an adviser is activated and informed of the new domain of this variable. When the adviser detects that propagation can happen, it will activate the propagator.

Both techniques have their advantages and disadvantages: classic propagators are conceptually simpler but need to iterate over all its variables when activated; advisers are more fine-grained but require more bookkeeping. We implemented the coverage constraint using both techniques, and compare them in the left part of Table 3.9. Using advisers requires more memory but reduces the overall amount of propagations. The runtimes also decrease.

Coverage Constraint: Clauses vs Sums As we pointed out in Property 3.2 on page 40, the coverage constraint can be expressed in two equivalent ways: using reified sums or using reified clauses. Both options are evaluated in Table 3.9. Overall, we find that the formulation using clauses performs best.

Value ordering For boolean decision variables, two value ordering heuristics are meaningful: selecting the minimum value (0) or selecting the maximum value (1) first. A comparison can be found in Table 3.8, where the maximum degree variable ordering is used.

The results are surprising: using the maximum value heuristic leads to more propagation and longer run times. This is counter-intuitive: the search tree is equally large in both cases and because the complete tree is searched, the

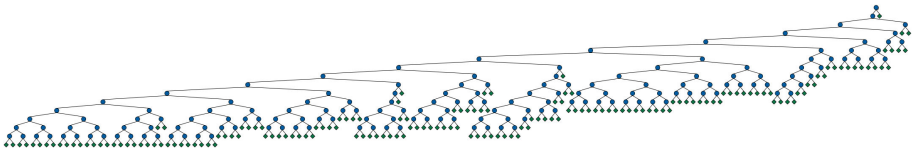


Figure 3.9: Search tree for the first 35 variables of the mushroom dataset. Every blue circle is a branch point over an item, every green diamond is a solution. A branch to the left assigned 0 to the item of that branch point, a branch to the right assigned 1.

total amount of propagation should be identical too. The explanation can be found in how the Gecode system stores intermediate states during the search. Gecode uses a technique called copying and recomputation [Schulte 2002]. In this technique, some nodes, but not necessarily all nodes, in the depth-first search tree are copied and stored. To backtrack, one retrieves the latest copied node and recomputes the propagations using the assignments leading to the desired node. This can save memory consumption and runtime for large problems [Schulte 2002]. The amount of copying/recomputation is set by the *copy distance* parameter. In Gecode, the default is 8, meaning that a new copy is made every 8 nodes.

When we consider a search tree using the minimum value first heuristic for our models (see Figure 3.9), we see that all variables are set to zero first, creating one long branch. The copied nodes in this branch are reused throughout the rest of the search. When using the maximum value heuristic, more propagation is possible and shorter branches are explored first. Consequently, less nodes are copied, and a great deal of recomputation needs to be done in each of the short branches. In our experiments this results in increased overhead.

Table 3.10 compares two values of the copy distance parameter, and how this influences the value ordering heuristic. With a distance of 0, every node in the search tree is copied. This results in a smaller amount of propagation compared to a distance of 8, independent of the value ordering heuristic used. Interestingly, the amount of runtime is also decreased compared to a larger copy distance. Using the maximum value first heuristic is about as fast as the minimum value heuristic, but needs significantly less memory. For our application it is thus faster and less memory intensive to clone every node in the search tree and choose the maximum value first.

	boolean sum			**boolean sum, advisers**			**clause, advisers**		
	mem.	props.	time	mem.	props.	time	mem.	props.	time
soybean	1796	1058238	0.5	2500	799791	0.5	1860	799791	0.5
splice-1	123279	6098820	68	237071	3188139	54	124431	3188139	49
anneal	2500	121495848	74	2500	121495848	73	2116	121495848	73
mushroom	30852	520928196	387	47172	344377153	372	31236	344377153	365

Table 3.9: Comparison in propagations, peak memory and time (in seconds) of formulating the coverage constraints using: the new reified sum constraint, the advisor version of the new reified sum constraint and the clause constraint. 'mem.' = peak memory in kilobytes, 'props.' = number of propagations, 'time' = run time in seconds.

	minimum, c-d 8			**minimum, c-d 0**			**maximum, c-d 0**		
	mem	props	time	mem	props	time	mem	props	time
soybean	1860	137666	0.2	7626	64823	0.2	1796	64823	0.2
splice-1	122511	2602069	41	911863	1822064	24	16328	1822064	23
anneal	1796	577719	18	4231	224555	19	2501	224555	19
mushroom	26244	6232932	48	148173	2853248	43	20229	2853248	43

Table 3.10: Comparison of peak memory, propagations and time (in seconds) using the minimum or maximum value ordering heuristics. The copy distance is either the default (c-d 8) or zero (c-d 0).

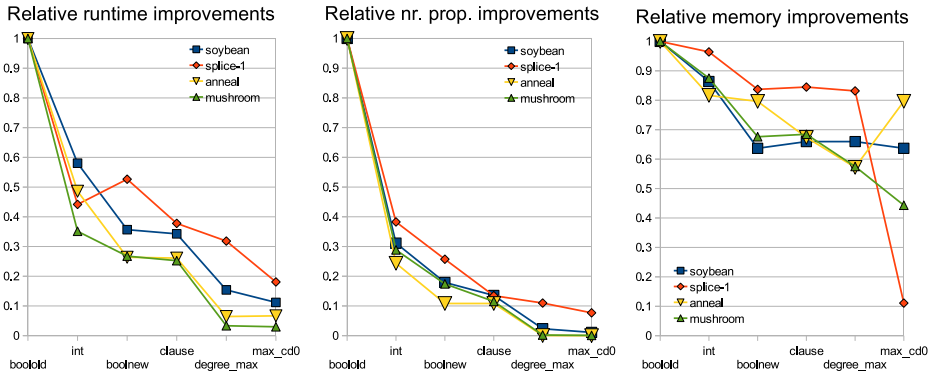


Figure 3.10: Relative improvements from applying several heuristics.

Summary An overview of the relative improvements of each step can be found in Figure 3.10. Overall, we see that even though our initial model –using reified constraints– could be specified straightforwardly, the scalability of the approach is highly depended on making the right low-level decisions, as discussed in this section. Only this modeling process as a whole can make the CP-based approach competitive with the current highly optimized systems for constraint-based itemset mining.

Making these decisions requires knowledge about how the underlying solver works as well as the experimental comparison of different options. However, general guidelines can often be drawn. In our case, the choices of propagators and heuristics explained in this section have shown to be valid for all pattern mining problems in this thesis.

Chapter 4

Integrating Constraint Programming and Itemset Mining

This chapter[1] investigates the performance gap between specialised miners and generic constraint solvers in more detail. It does this for the same problem and constraint formulations of the previous chapter. We present complexity results and experimentally demonstrate that an integrated solver can largely close the performance gap.

4.1 Introduction

The focus of most studies on pattern mining is on efficiently computing the set of solutions, given a specific set of constraints. The frequent itemset mining implementation challenge [Goethals and Zaki 2004] was organized to fairly compare the runtime efficiency and memory consumption of different algorithms that were proposed over the years. An overwhelming number of aspects have been studied, ranging from algorithmic aspects, such as breadth-first or depth-first search; data structures, such as using row-oriented (horizontal), column-oriented (vertical), or tree-based representations; as well

[1] Based on the paper "Integrating Constraint Programming and Itemset Mining" [Nijssen and Guns 2010].

as implementation aspects, such as the most efficient way to represent sets and count their cardinality.

An issue which received considerably less attention is how *generally applicable* the algorithms and their proposed optimizations are towards new constraints and tasks. Indeed, mining algorithms are usually limited to a small number of primitives that are hard-coded in the algorithm; no support is provided for adding constraints other than post-processing the patterns. We proposed an alternative approach in Chapter 3, which relies on the use of existing *constraint programming* systems. We showed that several well-known pattern mining problems can be modeled using constraint programming primitives. At the same time, however, we found that state-of-the-art CP systems often perform an order of magnitude worse than well-known itemset mining algorithms on popular itemset mining tasks such as closed and frequent itemset mining. This performance gap could become troublesome for very large datasets, as well as when using very low frequency thresholds. To make constraint programming an all-round alternative to specialized algorithms, we believe that it is essential that the performance gap is reduced.

In this chapter we study this problem. We perform an analysis which shows that existing CP systems represent data in a highly redundant way and that propagation is more often performed than necessary. To address this problem, we propose several key changes in CP systems: (1) we propose to use representations of data common in data mining in CP; (2) we propose that propagators share these representations; (3) we propose a mechanism through which propagators can share inferred constraints with each other; this allows to eliminate redundant computations by the propagators.

The resulting system maintains the principles of CP systems while being far more efficient than out-of-the-box constraint solvers. In particular, we will show that our CP system also achieves *polynomial delay* on mining problems that were only recently shown to have such complexity [Boley et al. 2007; Arimura and Uno 2009]; we show that CP provides an alternative framework for deriving algorithms of low computational complexity and illustrate this on a problem in *graph mining*. This observation is interesting as it shows that CP can be used as fundamental methodology for reasoning about data mining problems, including their computational complexity.

The chapter is organized as follows. In Section 4.2 we study the inner workings of frequent itemset mining and constraint programming in more detail. In Section 4.3 we analyse the bottlenecks of current CP solvers by comparing them with traditional mining algorithms. In Section 4.4 we present our integrated approach to solving these problems. Section 4.5 analyses the theoretical complexity of our system while Section 4.6 studies it experimentally; Section 4.7

concludes.

4.2 Existing Methods

Before studying how to integrate itemset miners and constraint programming systems, let us analyse how they work in more detail.

4.2.1 Frequent Itemset Mining

Existing Algorithms Many algorithms have been proposed for the frequent itemset mining problem. We discussed basic itemset mining algorithms in Section 2.1.1. We here focus on three major algorithmic choices: the search strategy, the representation of the data and the representation of the sets.

Search strategy: the most well-known algorithm is the breadth-first APRIORI algorithm [Agrawal et al. 1996]; an alternative search strategy is depth-first search. The latter strategy is taken in most of the recent algorithms for reasons of memory efficiency.

Representation of the data: the itemset database $\mathcal{D}$ can be represented in three equivalent ways:

- as a binary matrix of size $n \times m$, having entries $\mathcal{D}_{ti}$;
- as a bag of n itemsets $\mathcal{D}_t$, each of which represents a transaction with $\mathcal{D}_t = \{i \in \mathcal{I} \mid \mathcal{D}_{ti} = 1\}$. This is referred to as a *horizontal* representation of the binary matrix.
- as a bag of m tid-lists $\mathcal{D}_i^\mathsf{T}$ (one for each item), where $\mathcal{D}^\mathsf{T}$ is the transpose of matrix $\mathcal{D}$. Each $\mathcal{D}_i^\mathsf{T}$ contains a set of transaction identifiers such that $\mathcal{D}_i^\mathsf{T} = \{t \in \mathcal{S} \mid \mathcal{D}_{ti} = 1\}$. This is referred to as a *vertical representation* of the binary matrix.

More complex representations also exist: FP-Growth [Han et al. 2000], for instance, represents the itemset database more compactly in a prefix-tree. We do not consider this representation here.

Representation of sets: sets of tids such as T and $\mathcal{D}_i^\mathsf{T}$ can be represented in multiple ways. One is a *sparse representation*, consisting of a list of transaction identifiers included in the set (which is called a *positive representation*) or not included in the set (which is called a *negative representation*); the size of this representation changes as the number

of elements in the set changes. The other is a *dense representation*, in which case the set is represented as a boolean array. Large sets may require less space in this representation.

In Section 2.1.1 we presented a generic depth-first search algorithm and the concept of a projected database in which irrelevant items and transactions are removed during search. A representative state-of-the-art depth-first algorithm is the ECLAT algorithm [Zaki et al. 1997], which uses a vertical representation of the data (tid-lists); it can be implemented both for sparse and dense representations. The main observation used in ECLAT is the following:

$$\varphi(I) = \bigcap_{i \in I} \varphi(\{i\}) = \bigcap_{i \in I} \mathcal{D}_i^{\mathsf{T}}. \tag{4.1}$$

In other words, the cover of an itemset I equals the intersection of the covers of the individual items, which in turn equals the intersection of the tid-lists (vertical representations) of those items. Hence, the cover of an itemset can be incrementally calculated during search by intersecting it with the tid-lists of the remaining items.

This allows for a search strategy depicted in Algorithm 4.1. The initial call of this algorithm is for $I = \emptyset$, $T = \mathcal{S}$ and $I_{pos} = \mathcal{I}$. The depth-first search strategy of Algorithm 4.1 is based on the following properties:

1. To avoid an itemset I from being generated multiple times, an order is imposed on the items. We do not add items to an itemset I which are lower or equal to the highest item already in the set (line 3).

2. When adding an item to an itemset, we can calculate the tid-set of the resulting itemset incrementally from the tid-set of the original itemset as follows: $\varphi(I \cup \{i\}) = \varphi(I) \cap \mathcal{D}_i^{\mathsf{T}}$ (line 4). This corresponds to a column in the newly projected database of this itemset.

3. If an itemset I is frequent, but itemset $I \cup \{i\}$ is infrequent, all sets $J \supseteq I \cup \{i\}$ are also infrequent. In I'_{pos} we maintain those items which can be added to I and yield a frequent itemset (line 5).

These properties correspond to lines 2, 5 and 6 in Algorithm 2.1 on page 15, regarding the order of items and the transactions/items in the projected database.

Algorithm 4.1 Eclat(I,T,I_{pos},$\mathcal{D}$)

```
1: Output I
2: I'_pos = ∅
3: for all i ∈ I_pos, i > max(I) do
4:     if |T ∩ D_i^T| ≥ θ then
5:         I'_pos := I'_pos ∪ {i}
6:     end if
7: end for
8: for all i ∈ I'_pos do
9:     Eclat(I ∪ {i},T ∩ D_i^T,I'_pos,D)
10: end for
```

4.2.2 Constraint Programming

Existing Algorithms Most constraint programming systems perform depth-first search. We have explained the basic principles of a generic algorithm in Section 2.2.1 on page 25. The key concept used to speed-up the search is constraint propagation, which reduces the domains of variables such that the domain remains consistent.

There are many different CP systems. We discussed a number of design decisions in Section 2.2.4 of the background chapter. In this chapter, the following differences are considered in more detail:

Types of variables: many solvers implement integer variables, of which the domains can be represented in two ways: representing every element in the domain separately or saving only the *bounds* of the domain, namely the minimum and maximum value the variable can still take.

Supported constraints: related to the types of variables supported, the supported constraints determine the problems that can be modeled and solved by a solver.

Propagator activation: when the domain of a variable changes, the propagators involving this variable need to be activated. Activation can be done at different levels of granularity, differing in how much information is supplied to the propagator (see Section 2.2.4).

4.3 Comparison of Methods

We briefly summarize the CP formulation presented in Chapter 3. We then study in more detail how CP solvers and itemset mining algorithms differ in the properties introduced in the previous section. In the next section we will use the best of both worlds to develop a new algorithm.

Problem Formulation To model itemset mining in a CP system, we use two sets of boolean variables:

- a variable I_i for each item i, which is 1 if the item is included in the solution and 0 otherwise. The vector $I = (I_1, \ldots, I_m)$ represents an itemset.
- a variable T_t for each transaction t, which is 1 if the transaction is in the solution and 0 otherwise. The vector $T = (T_1, \ldots, T_n)$ represents a tid-set.

A solution hence represents one itemset I with corresponding tid-set T. To find all frequent itemsets, we need to iterate over all solutions satisfying the following constraints:

$$\forall t \in \mathcal{S} : T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0 \qquad (Coverage) \tag{4.2}$$

$$\forall i \in \mathcal{I} : I_i = 1 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta \qquad (Min.\ frequency) \tag{4.3}$$

Other well-known itemset mining problems introduced in Chapter 2 can be formalized in a similar way. An overview of constraints is provided in the first two columns of Table 4.1. The general type of constraint that we used is of the following form:

$$\forall x \in \mathcal{X} : X_x = b \leftrightarrow \sum_{y \in \mathcal{Y}} Y_y d_{xy} \lesseqgtr \theta. \tag{4.4}$$

where $b, d_{xy} \in \{0, 1\}$ are constants, $\lesseqgtr \in \{\leq, =, \geq\}$ are comparison operators, and each X_x and Y_y is a boolean variable in the CP model.

CP compared to itemset mining algorithms We use the Gecode constraint programming system [Schulte and Stuckey 2008] as a representative CP system. It is one of the most prominent open and extendible constraint programming systems, and is known to be very efficient. We will again use the ECLAT

Constraint	Reified sums	Matrix notation
Coverage	$\forall t \in \mathcal{S} : T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0$	$\vec{T} \leq \mathbf{1}_{=0}((1-\mathcal{D})\vec{I})$❶ & $\vec{T} \geq \mathbf{1}_{=0}((1-\mathcal{D})\vec{I})$❷
Min. frequency	$\forall i \in \mathcal{I} : I_i = 1 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta$	$\vec{I} \leq \mathbf{1}_{\geq\theta}(\mathcal{D}^\intercal \vec{T})$ ❸
Closedness	$\forall i \in \mathcal{I} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{S}} T_t(1 - \mathcal{D}_{ti}) = 0$	$\vec{I} \leq \mathbf{1}_{=0}((1-\mathcal{D})\vec{T})$ ❹ & $\vec{I} \geq \mathbf{1}_{=0}((1-\mathcal{D})\vec{T})$ ❺
Maximality	$\forall i \in \mathcal{I} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta$	$\vec{I} \geq \mathbf{1}_{\geq\theta}(\mathcal{D}^\intercal \vec{T})$ ❻ & ❸
Min. size	$\forall t \in \mathcal{S} : T_t = 1 \rightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} \geq \theta'$	$\vec{T} \leq \mathbf{1}_{\geq\theta'}(\mathcal{D}\vec{I})$ ❼
Min. cost	$\forall t \in \mathcal{S} : T_t = 1 \rightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} c_i \geq \theta'$	$\vec{T} \leq \mathbf{1}_{\geq\theta'}((\mathcal{D}C)\vec{I})$ ❼

Table 4.1: Formalizations of common itemset mining primitives in CP.

algorithm as representative itemset mining algorithm. To recapitulate from Section 4.2.1, key choices for itemset mining algorithms are the search strategy and the representation of the data.

As a reminder, in both Gecode and Eclat search proceeds depth-first, but the search tree in Gecode is binary. In Eclat every node in the search tree corresponds to one itemset, and the search only adds items to the set (Figure 2.3).

The database is not explicitly represented in the CP model; instead rows and columns are spread over the constraints. Studying constraints (4.2) and (4.3) in detail, we observe that for every transaction there is a reified summation constraint containing all the items not in this transaction; for every item there is a constraint containing all transactions containing this item. Furthermore, to activate a propagator when the domain of a variable is changed, for every item and transaction variable there is a list containing propagators depending on it. Overall, the data is hence stored 4 times, both in horizontal and vertical representations, and in positive and negative representations. In Eclat a database is only stored in one representation, which can be tuned to the type of data at hand (for instance, using positive or negative representations). Although in terms of worst case space complexity the performance is hence of the same order, the amount of overhead in the CP system is much higher.

Finally, we wish to point out that in CP, all constraints are independent. The reified formulation of the frequency constraint checks for every item whether it is still frequent, even if it is already included in the itemset. As the reified constraints do not share information with each other, this redundant computation cannot be avoided.

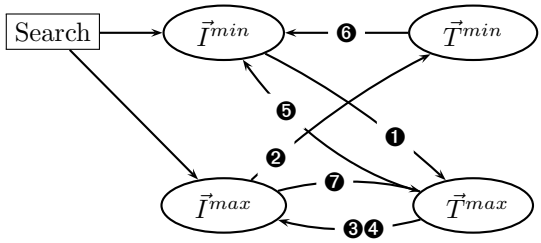


Figure 4.1: Propagators for itemset mining as functions between bounds of variables; the search is assumed to branch over items.

Itemset mining compared to CP solvers In Section 4.2.2 we identified types of variables, supported constraints and propagator activation as key differences between solvers. In Gecode, every item and every transaction is represented by an individual boolean variable. The representation of variables is hence dense. The representation of the constraints, on the other hand, is sparse. Itemset mining algorithms like Eclat are more flexible; the choice of representation for sets is often left open. Considering supported constraints, the original Eclat algorithm deals mainly with one type of constraint: the minimum frequency constraint. For other types of constraints, such as closedness constraints, the algorithm needs to be (and has been) extended. Gecode on the other hand supports many different constraints out-of-the-box and imposes no restriction on which ones to use or how to combine them.

To compare propagator activation in Eclat and Gecode, let us consider when the change of a variable is propagated. Let us denote by $\vec{I}^{min}$ the lower-bounds of the itemset variables I. Similar reasoning holds for $\vec{I}^{max}$, $\vec{T}^{min}$ and $\vec{T}^{max}$. Figure 4.1 visualizes the propagation that is necessary for the constraints presented in Table 4.1. An arrow indicates when a certain propagator needs to be activated. For example, when an itemset i is set to one ($I_i^{min} = 1$), we need to activate the coverage propagator. In the case of frequent itemset mining, only propagations ❶ and ❸ are strictly needed; propagation ❷ is useless as the lower-bound of the transaction variables (representing transactions that are certainly covered) is never used later on. Indeed Eclat only performs propagation ❶ and ❸. However, Gecode activates propagators more often (to no possible beneficial effect in itemset mining). The reason is that in Gecode a propagator is activated whenever a variable's domain is changed, independent of whether the upper-bound or lower-bound changed (coarse-grained activation). Hence, if an item's domain changes after frequency propagation, the coverage propagator will be triggered again already due to the presence of constraint ❶ [2].

[2]If we would enable propagation ❷, the domain of T may change; this would lead to even further (useless) propagation towards the item variables.

Conclusion Overall, we know that the search procedures are similar at a high level, but we can observe that the constraint programming system faces significant overhead in data representation, data maintenance, and constraint activation.

4.4 An Integrated Approach

We propose an integrated approach that builds on the generality of CP, but aims to avoid the overhead of existing systems. Our algorithm implements the basic constraint search algorithm, augmented with the following features: (1) a boolean vector as basic variable type; (2) support for multiple matrix representations; (3) a general matrix constraint that encompasses itemset mining constraints; (4) an auxiliary store in which facts are shared; (5) efficient propagators for matrix constraints. To the best of our knowledge, there is currently no CP system that implements these features. In particular, our use of an auxiliary store that all constraints can access is uncommon in existing CP systems and mainly motivated by our itemset mining requirements.

Boolean vectors Our first choice is to use boolean vectors as basic variable types; such a vector can be interpreted as a subset of a finite set of elements. Constraints will be posted on the set of items I and the set of transactions T as a whole. In our system, a finite domain set is a boolean vector where the ith bit indicates if element number i is part of the set. The use of set variables in constraint programming is not new [Puget 1992; Gervet 1997]. In the boolean vector representation we use, the domain of a boolean vector B is represented by its bounds which are stored in two boolean vectors B^{min} and B^{max}. This is similar to the representation used in the data mining system DualMiner [Bucila et al. 2003]. Search branches over individual booleans in the vectors.

Data representation When posting constraints, we support multiple matrix representations, such as vertical, horizontal, positive and negative representations. Constraints will operate on all representations, but more efficient propagators will be devised for some representations. Note that certain matrices can be seen as *views* on other matrices: for instance, $\mathcal{D}^{\mathsf{T}}$ is a *horizontal* view on a *vertically* represented matrix $\mathcal{D}$. Maintaining and using such views globally avoids that different constraints need to maintain their own representations of the data, and hence reduces the amount of redundancy.

$$\mathcal{D} \quad \vec{T} \quad \mathcal{D}^\intercal \cdot \vec{T} \quad \mathbf{1}_{\geq 2}(\mathcal{D}^\intercal \cdot \vec{T}) \quad \vec{I} \leq \mathbf{1}_{\geq 2}(\mathcal{D}^\intercal \cdot \vec{T})$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Figure 4.2: Stepwise example of the matrix constraint $\vec{I} \leq \mathbf{1}_{\geq 2}(\mathcal{D}^\intercal \cdot \vec{T})$ for a matrix $\mathcal{D}$ with 3 items and 5 transactions. Note in the last column that the example constraint is violated because of the second component of vector $\vec{I}$.

General matrix constraint We now reformulate the reified summation constraint, $\forall x \in \mathcal{X} : X_x = b \leftrightarrow \sum_{y \in \mathcal{Y}} Y_y d_{xy} \lesseqgtr \theta$ (Equation (4.4)), using boolean vectors and matrix operations. The general form of the reified matrix constraint is the following:

$$\vec{X} \lesseqgtr \mathbf{1}_{\gtreqless \theta}(\mathcal{A} \cdot \vec{Y}); \tag{4.5}$$

where $\lesseqgtr$ and $\gtreqless$ can be replaced by either $\geq$ or $\leq$; $\vec{X}$ and $\vec{Y}$ are boolean column vectors; $\mathcal{A}$ is a matrix; $\cdot$ denotes the traditional matrix product; $\mathbf{1}_{\geq \theta}$ is an *indicator function* which is applied element-wise to vectors: in this case, if the ith component of $\mathcal{A} \cdot \vec{Y}$ exceeds threshold θ, the ith component of the result is 1; otherwise it is 0. In comparing two vectors $\vec{X}_1 \geq \vec{X}_2$ it is required that every component of $\vec{X}_1$ is not lower than the corresponding component of $\vec{Y}_2$.

For instance, the minimum frequency constraint can be formalized as an instantiation of the general matrix constraint as follows:

$$\vec{I} \leq \mathbf{1}_{\geq \theta}(\mathcal{D}^\intercal \cdot \vec{T}). \tag{4.6}$$

Figure 4.2 shows an example for a given $\mathcal{D}, \vec{T}$ and $\vec{I}$. The constraint expresses that only if the vector product of the transaction vector $\vec{T}$ with column vector i of the data exceeds threshold θ, the ith component of vector $\vec{I}$ can be 1, and this for all components of $\vec{I}$. In the example, the constraint is violated because this is not the case for the second component of $\vec{I}$; the indicator function shows that it is not frequent in the given transactions for the given threshold.

We can reformulate many itemset mining constraints using this notation, as shown in Table 4.1. In this notation we assume that $(1 - \mathcal{D})$ yields a matrix $\mathcal{D}'$ in which $\mathcal{D}'_{ij} = 1 - \mathcal{D}_{ij}$; furthermore, C represents a diagonal matrix in which the costs of items are on the diagonal.

As with any CP system, other constraints and even other variable types can be added; for the typical itemset mining settings discussed in this chapter no other constraints are needed.

Propagation framework A propagator for the general matrix constraint can be thought of as a function taking boolean vectors (representing bounds) as input, and producing a boolean vector (representing another bound) as output. For example, a propagator for the frequency constraint in Equation 4.6 is essentially a function that derives from $\vec{T}^{max}$ the values of $\vec{I}^{max}$, e.g. in the example in Figure 4.2 the $\vec{T}$ would be a $\vec{T}^{max}$ and $\vec{I}$ an $\vec{I}^{max}$ and the propagator would propagate that the second component of $\vec{I}^{max}$ has to be zero.

In a fine-grained activation scheme, the constraint would only have to be activated if the upper-bound of $\vec{T}$ changes. Figure 4.1 lists all these relationships for the constraints in Table 4.1. Our system implements such fine-grained propagator activation, we activate the propagator of a constraint only if the bound(s) it takes as input have changed, avoiding useless propagation calls.

Auxiliary store The main idea is to store a set of simple, redundant constraints that are known to hold, given the constraints and the current variable assignments, for this search node and all of its children. These constraints are of the form $\vec{X} \subseteq \mathcal{A}_i$, where $\subseteq$ may be replaced by other set operators, $\vec{X}$ is a vector variable in the constraint program, and $\mathcal{A}_i$ a column in the data. Propagators may insert such *satisfied* constraints in the store, and may query the store later on to avoid the data access and subsequent computation needed to determine the truth value of the constraint. The store is maintained dynamically during search; stored constraints are removed when backtracking over the search nodes that introduced them. In the worst case, the size of the store is $O(m + n)$ with $m = |\mathcal{D}|$ and $n = |\mathcal{I}|$, namely if for every row and column of the matrix a constraint is stored.

Efficient Propagators The general propagator for one particular choice for the inequality directions in the matrix constraint $\vec{X} \leq \mathbf{1}_{\geq\theta}(\mathcal{A} \cdot \vec{Y})$, is the following:

$$\vec{X}^{max} \leftarrow \min(\vec{X}^{max}, \mathbf{1}_{\geq\theta}(\mathcal{A} \cdot \vec{Y}^{max})), \tag{4.7}$$

where we presume $\mathcal{A}$ only contains non-negative entries. This propagator takes as input vector $\vec{Y}^{max}$ and computes $\vec{X}^{max}$ as output, possibly exploiting the value $\vec{X}^{max}$ currently has. Upon completion of the propagator, it should be checked if $\vec{X}^{min} \leq \vec{X}^{max}$; otherwise a contradiction is found and the search will backtrack.

We provide the specialized propagators of the coverage and frequency constraint on a vertical representation of the data below.

Coverage: The coverage constraint is $\vec{T} \leq \mathbf{1}_{\leq 0}(\mathcal{A} \cdot \vec{I})$, where $\mathcal{A} = 1 - \mathcal{D}$, i.e., a binary matrix represented vertically in a negative representation where $\mathcal{D}$ is the positive representation. The propagator should evaluate $\vec{T}^{max} \leftarrow \min(\vec{T}^{max}, \mathbf{1}_{\leq 0}(\mathcal{A} \cdot \vec{I}^{min}))$. One individual $\vec{T}_t^{max}$ is set to 0 if $\mathbf{1}_{\leq 0}(\mathcal{A} \cdot \vec{I}^{min})_t = 0$, which corresponds to $\sum_{i \in \mathcal{I}} \mathcal{A}_{it} * \vec{I}_i^{min} \leq 0$ being false, or equivalently $\sum_{i \in \mathcal{I}} (1 - \mathcal{D}_{ti}) * \vec{I}_i^{min} \leq 0$ being false. This corresponds to propagating the following part of the coverage constraint in Equation 4.2: $\forall t \in \mathcal{S} : T_t^{max} = 1 \rightarrow \sum_{i \in \mathcal{I}^{min}} I_i(1 - \mathcal{D}_{ti}) = 0$. Another way of looking at this propagator is that it enforces $T^{max} \leftarrow T^{max} \cap \varphi(I^{min})$, namely that a transaction T_t^{max} is set to 0 if it is not covered by I^{min}. Taking inspiration from traditional itemset mining (Equation (4.1)), this propagator can also be evaluated as follows:

$$T^{max} \leftarrow T^{max} \cap \left(\bigcap_{i \in I^{min}} \mathcal{D}_i^{\mathsf{T}} \right). \tag{4.8}$$

Observe that the latter propagator uses the vertical matrix representation $\mathcal{D}^{\mathsf{T}}$ directly, without needing to compute the negative matrix $\mathcal{A}$. The propagator skips columns which are not currently included in the itemset ($i \in I^{min}$). After having executed this propagator, we know for a number of items $i \in \mathcal{I}$ that $T \subseteq \mathcal{D}_i^{\mathsf{T}}$. We will store this knowledge in the auxiliary constraint store to avoid recomputing it. The propagator is given in Algorithm 4.2.

Frequency: The minimum frequency constraint is $\vec{I} \leq \mathbf{1}_{\geq \theta}(\mathcal{A} \cdot \vec{T})$. If $\mathcal{A} = \mathcal{D}^{\mathsf{T}}$ for the vertically represented matrix $\mathcal{D}$ then $\mathcal{A}$ is the same binary matrix that can be seen as a horizontal representation of $\mathcal{D}^{\mathsf{T}}$. The propagator should evaluate:

$$\vec{I}^{max} \leftarrow \min(\vec{I}^{max}, \mathbf{1}_{\geq \theta}(\mathcal{A} \cdot \vec{T}^{max})).$$

We can evaluate this constraint by computing for every $i \in I^{max}$ the size of the vector product $|\mathcal{D}_i^{\mathsf{T}} \cdot T^{max}|$. If this number is lower than θ, we can set $I_i^{max} = 0$.

We speed up this computation using the auxiliary constraint store. If for an item i we know that $T^{max} \subseteq \mathcal{D}_i^{\mathsf{T}}$, then $|\mathcal{D}_i^{\mathsf{T}} \cdot T^{max}| = |T^{max}|$ so we only have to compute $|T^{max}| \geq \theta$. The propagator can use the auxiliary store for this; see Algorithm 4.3. This is the same information that the coverage constraint uses; hence it can happen at some point during the search that the propagators do not need to access the data any longer. Storing and maintaining the additional information in the auxiliary constraint store will require additional $O(m + n)$ time and space for each itemset (as we need to store an additional transaction set or itemset for each itemset); the potential gain in performance in the frequency propagators is $O(mn)$, as we can potentially avoid having to consider all elements of the data again.

Algorithm 4.2 PropCoverage($\vec{I}^{min}$,D,$\vec{T}^{max}$)

1: **for all** $i \in \vec{I}^{min}$ **do**
2: **if** $(\vec{T}^{max} \subseteq \mathcal{D}_i^{\mathsf{T}}) \notin$ store **then**
3: $\vec{T}^{max} := \vec{T}^{max} \cap \mathcal{D}_i^{\mathsf{T}}$
4: Store $(\vec{T}^{max} \subseteq \mathcal{D}_i^{\mathsf{T}})$
5: **end if**
6: **end for**

Algorithm 4.3 PropFrequency($\vec{T}^{max}$,D,$\vec{I}^{max}$)

1: $F := |\vec{T}^{max}|$
2: **for all** $i \in \vec{I}^{max}$ **do**
3: **if** $(\vec{T}^{max} \subseteq \mathcal{D}_i^{\mathsf{T}}) \notin$ store **then**
4: $F' := |\mathcal{D}_i^{\mathsf{T}} \vec{T}^{max}|$
5: **if** $F' < \theta$ **then** $I_i^{max} := 0$
6: **if** $F' = F$ **then** Store $(\vec{T}^{max} \subseteq \mathcal{D}_i^{\mathsf{T}})$
7: **end if**
8: **end for**

4.5 Complexity Analysis

In this section we study the complexity of our constraint programming system on itemset mining tasks. Assuming that propagators are evaluated in polynomial space, the space complexity of the approach is polynomial, as the search proceeds depth-first and no information is passed from one branch of the search tree to another. In general, polynomial *time* complexity is not to be expected since CP systems can be used to solve NP complete problems.

For certain tasks, however, we can prove *polynomial delay* complexity, i.e. before and after each solution is printed, the computation time is polynomial in the size of the input. In particular, the CP-based approach provides an alternative to the recent theory of accessible set systems [Arimura and Uno 2009], which was used to prove that certain closed itemset mining problems can be solved with polynomial delay.

Consider the Constraint-Search(D) algorithm in Algorithm 2.2 on page 25; we observe the following regarding the complexity of the Propagate(D) procedure on line 1. If k is the sum of the lengths of the boolean vector variables in the constraint specification of a problem, a fixed point will be reached in $O(k)$ iterations, as this is the maximum number of bits that may change in all iterations of a propagation phase together. If each propagator can be evaluated in polynomial time, then a call to Propagate(D) will execute in polynomial time. Using this we can prove the following.

Theorem 4. *If in Algorithm 2.2 all propagators and the variable selection function f can be evaluated in time polynomial in the input and each failing leaf in the binary search tree has a non-failing sibling, then solutions to the constraint program will be listed with polynomial delay by that algorithm.*

Proof. Essentially we need to show that the number of failing leaves between two successive non-failing leaves of the search tree is polynomial in the size of the input. Assume we have an internal node, then given our assumption, independent of the order in which we consider its children, we will reach a leaf after considering a path of at most k nodes in the depth-first search. Either this leaf is a non-failing leaf, or its sibling is non-failing. In the latter case, we can again consider a path of at most k nodes from this node, and reach a leaf that is non-failing or that has a non-failing sibling. Given that k is the maximum depth of the search tree, a non-failing leaf will be reached after visiting $O(k)$ nodes. Consider the successive non-failing leaf, we will need at most $O(k)$ steps to reach the ancestor for which we need to consider the other child. From the ancestor we will reach the non-failing leaf after considering $O(k)$ nodes. □

From this theorem it follows that frequent and frequent closed itemsets can be listed with polynomial delay: consider frequent itemset mining, which only needs propagations ❶ and ❸ (see Section 4.3). In Figure 4.1 we can see that propagation ❶ and ❸ will only happen when I^{min} or T^{max} changes. The search tree for frequent itemset mining is a binary tree where every node corresponds to an item and the two siblings represent the item being true or false. If the item is set to true, I^{min} will change, otherwise I^{max}. As no propagation will happen when I^{max} changes, every node in the search tree has a non-failing sibling, as required by the theorem.

The same holds for frequent closed itemset mining, which uses propagations ❶ and ❸ as well as ❹ and ❺. Neither of these depend on I^{max} (see Figure 4.1), hence the theorem holds for the same reasons as for the frequent itemset mining problem. The theorem does not hold for maximal frequent itemset mining. The maximality constraint adds propagation ❺, which in turn requires propagation ❷. As propagation ❷ is activated when I^{max} changes, the non-failing sibling condition of Theorem 4 is no longer met.

The same procedure can also be used for more complex itemset mining problems. We illustrate this for a simple graph mining problem introduced in [Boley et al. 2007], which illustrates at the same time how our approach can be extended to other mining problems. In addition to a traditional itemset database, in this problem setting a graph G is given in which items constitute nodes and edges connect nodes. We denote by $G[I]$ the subgraph of G that is induced by the itemset I, that is, the subgraph of G in which every item in I is a node and the edges are all edges in G connecting nodes also in $G[I]$.

In addition to the coverage and minimum frequency constraint, our graph mining problem consists of two more constraints, namely *connected*(I) and *connectedClosed*(I, T). An itemset I satisfies constraint *connected*(I) if $G[I]$

$\mathcal{D}$	Itemset
T_1	{A,B,C,D,F}
T_2	{A,B,C,D}
T_3	{A,D,E}
T_4	{A,B,C,D,E}
T_5	{B,C,F}
T_6	{D,E}

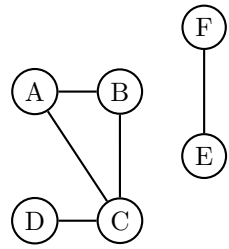


Figure 4.3: Example itemset database with accompanying graph

is connected. An itemset (I, T) satisfies constraint *connectedClosed*(I, T) iff I corresponds to a complete connected component in (the possibly unconnected) subgraph $G[\psi(T)]$, where $\psi(T) = \cap_{t \in T} \mathcal{D}_t$ consists of all items shared by the transactions in T.

Example 8. *Consider the database and graph in Figure 4.3, itemset* $\{A, C, D\}$ *satisfies* connected(I) *while itemset* $\{A, B, E\}$ *and* $\{A, D\}$ *do not. For* $I = \{A, B, C\}$ *covering* $T = \{T_1, T_2, T_4\}$*, we have that* $\psi(T) = \{A, B, C, D\}$ *where* I *is not a complete connected component in* $G[\{A, B, C, D\}]$*, hence* connectedClosed(I, T) *is not satisfied. For* $I = \{A, B, C, D\}$ *covering* $T = \{T_1, T_2, T_4\}$*, the constraint* connectedClosed(I, T) *is true.*

We can solve this problem in a similar way to itemset mining by using the same variables and the following propagators (propagator activation is visualised in Figure 4.4, similar to Figure 4.1 for regular itemset mining):

- the coverage and frequency constraint as in standard itemset mining, indicated by ❶ and ❸ in Figure 4.4;
- a propagator for *connected*(I), which takes $\vec{I}^{max}$ as input and $\vec{I}^{max}$ as output (❽); given one arbitrary variable for which $I_i^{min} = 1$, it determines the connected component induced by $\vec{I}^{max}$ that i is included in, and changes the domain $I_i^{max} = 0$ for all items i not in this component. It fails if this means that for some i': $I_{i'}^{min} > I_{i'}^{max}$;
- a propagator for *connectedClosed*(I, T), which for a given set $\vec{T}^{max}$ calculates $\psi(\vec{T}^{max})$ (❾), and sets $I_i^{min} = 1$ for all items in the component the current items in $\vec{I}^{min}$ are included in, possibly leading to a failure.

Additionally, we add a variable selection function f which as next item to split on selects an item which is connected in G to an item for which $I_i^{min} = 1$. The effect of this variable selection function is that propagator ❽ for *connected*(I)

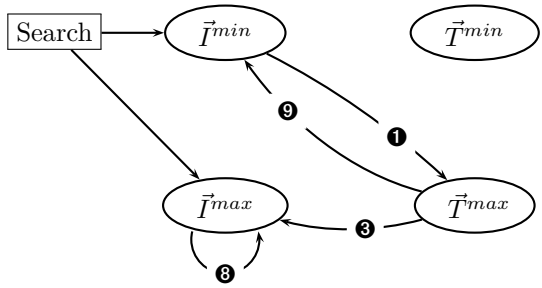


Figure 4.4: Propagators for the simple graph mining problem, as functions between bounds of variables. The search is assumed to branch over items.

will never lead to failure: items not in the same connected component will never be added to the set.

Regarding Theorem 4, Figure 4.4 shows that ❽ is the only propagator activated when changing I^{max}. Given that variable selection function f guarantees that ❽ never leads to failure, the branch of the search tree in which an $I_i^{max} = 0$ is set (the item is not in the set) will never fail. Hence, requirements for the theorem are fulfilled and the algorithm will enumerate the solutions with polynomial delay.

4.6 Experiments

In our experiments we aim to determine how much our specialized framework can contribute to reducing the performance gap between CP systems and itemset miners. As CP is particularly of interest when dealing with many different types of constraints, our main interest is achieving competitive behaviour across multiple tasks.

Fairly comparing itemset miners is however a daunting task, as witnessed by the large number of results from the FIMI competition [Goethals and Zaki 2004]. We report a limited number of settings here and refer to our website for more information, including the source code of our system: `http://dtai.cs.kuleuven.be/CP4IM/`.

We choose to restrict our comparison to the problems of frequent, closed and maximal itemset mining, as well as frequent itemset mining under a minimum size constraint. The reason is that there are a relatively large number of systems supporting all these tasks, some of which were initially developed for the FIMI competition, but have since been extended to deal with additional constraints. In particular, we used these systems:

Name	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{I}\rvert$	Density	# Patt. at $\theta = 25\%$	Source
T10I4D100K	100000	1000	1%	1	[1]
Splice	3190	290	21%	118	[2]
Mushroom	8124	120	19%	5546	[1]
Ionosphere	351	99	50%	184990186	[2]

Table 4.2: Characteristics of the data used in the experiments. [1]=[Goethals and Zaki 2004], [2]=[Nijssen et al. 2009].

DMCP our new constraint programming system, in which we always used a dense representation of sets and used a static order of items;

FIMCP the Gecode constraint programming system (used in the previous chapter);

PATTERNIST the engine underneath the ConQueSt constraint-based itemset mining system [Bonchi et al. 2009];

LCM2 and LCM5 the LCM algorithm [Uno et al. 2005] that was very successful in the FIMI competition. Version 2 is the most efficient while version 5 supports more constraints;

MAFIA An algorithm specialised in maximal itemset mining;

B_APRIORI, B_FPGrowth, B_ECLAT and B_ECLAT_NOR , the classic Apriori, Eclat and FPGrowth algorithms as implemented by C. Borgelt [Borgelt 2003]. **B_ECLAT** checks maximality of an itemset by searching for related itemsets in a repository of already determined maximal itemsets; **B_ECLAT_NOR** determines maximality of an itemset by checking the data.

Unless mentioned otherwise, the algorithms were run with default parameters; output was requested, but written to `/dev/null`. The algorithms were run on machines with Intel Q9550 processors and 8GB of RAM. Experiments were timed out after 1800s.

Characteristics of four diverse data sets are given in Table 4.2. Density is the relative number of ones in the binary matrix representing the data. T10I4D100K is a large dataset with very low density while Ionosphere is a small dataset with very high density. The splice and mushroom datasets differ mainly in their ratio of number of items/transactions. The number of patterns at a frequency threshold of 25% gives an indication of the necessity of constraints to keep the number of patterns low. The experiments serve to show the difference

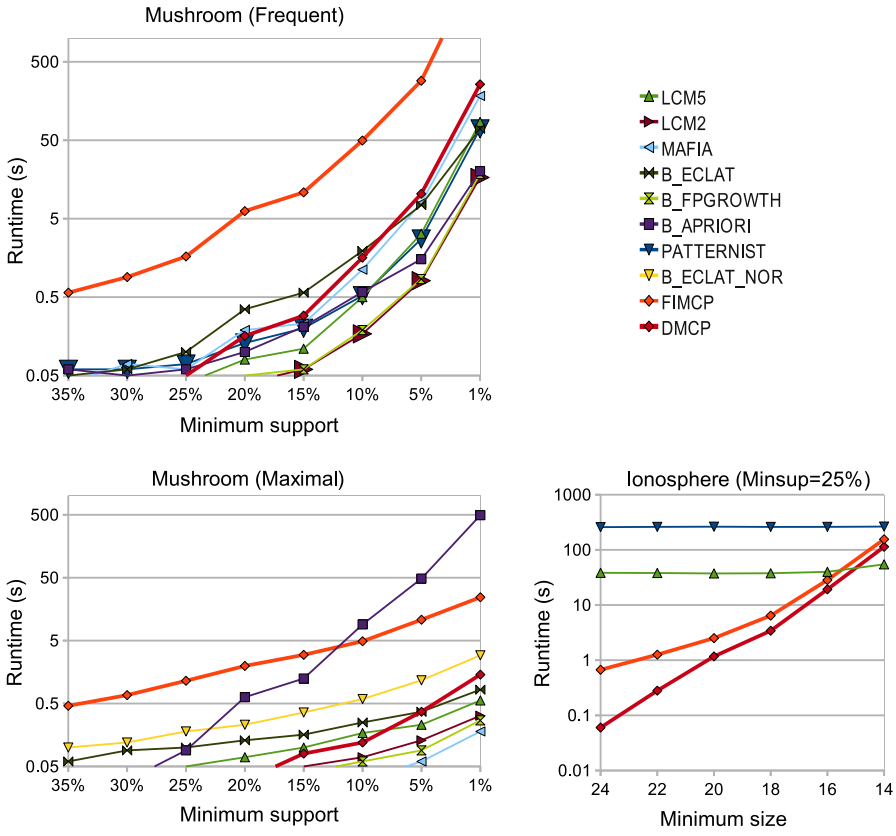


Figure 4.5: Run times of several algorithms on Mushroom and Ionosphere.

between FIMCP and DMCP in a number of traditional settings compared to traditional algorithms.

In the top-left of Figure 4.5, the results for mining all frequent itemsets on the popular mushroom dataset can be seen. The thick line on top is FIMCP, our Gecode-based implementation. The darker thick line represents the runtime of the introduced DMCP system. Its efficiency is comparable to that of specialised itemset miners. Compared to frequent itemset mining, maximal frequent itemset mining is a more constrained setting resulting in a smaller set of patterns; hence we would expect constraint programming to have an advantage for tasks like this. Indeed, in the bottom-left figure of Figure 4.5, we can observe that DMCP outperforms approaches such as B_APRIORI and

B_ECLAT_NOR, which in this case are filtering frequent itemsets[3]. The Gecode-based FIMCP does not sufficiently profit from the propagation to overcome its redundant data representation.

Investigating the use of constraints further, we apply a number of minimum size constraints on the Ionosphere data (bottom-right of Figure 4.5). This dataset contains so many frequent patterns that additional constraints are needed to find a reasonably sized set of patterns. A minimum size constraint is a common test case in constraint-based itemset mining, as it is monotone as opposed to the anti-monotone frequency constraint. Some itemset mining implementations, such as LCM, have added this constraint as an option to their systems. The results show that on this type of data the LCM system and, surprisingly, the PATTERNIST system that was designed for constraint-based mining, are (almost) not affected by this constraint. The CP based approaches, on the other hand, can effectively prune the search space as the constraint becomes more restrictive. DMCP is again superior compared to FIMCP.

Figure 4.6 (left) illustrates the behaviour of the systems on the very sparse T10I4D100K dataset. The Gecode-based FIMCP system suffers even more severely from its redundant representation of the data. When mining maximal frequent itemsets, the run time of most systems is very similar to the time needed to mine all frequent itemsets. The reason is that for sparse data, the number of maximal frequent itemsets is also similar to that of the total number of frequent itemsets. In particular in systems such as B_ECLAT and B_FPGROWTH, which use a repository to filter from the frequent itemsets, the maximality check gives minimal overhead. If we disable the repository (B_ECLAT_NOR), Eclat's performance is very similar to that of the DMCP system, as both systems are essentially filtering itemsets by checking maximality in the data. Similar behaviour is observed for the splice data, where the difference between closed and non-closed itemsets is small. In all figures it is clear that our specialised solver operates in the same ballpark as other itemset mining systems.

4.7 Conclusions

In this chapter we studied the differences in performance between general CP solvers and specialized mining algorithms. We focused our investigation on the representation of the data and the activation of propagators. This provided

[3]The difference between B_APRIORI and B_ECLAT_NOR is mainly caused by *perfect extension pruning* [Borgelt 2003], which we did not disable.

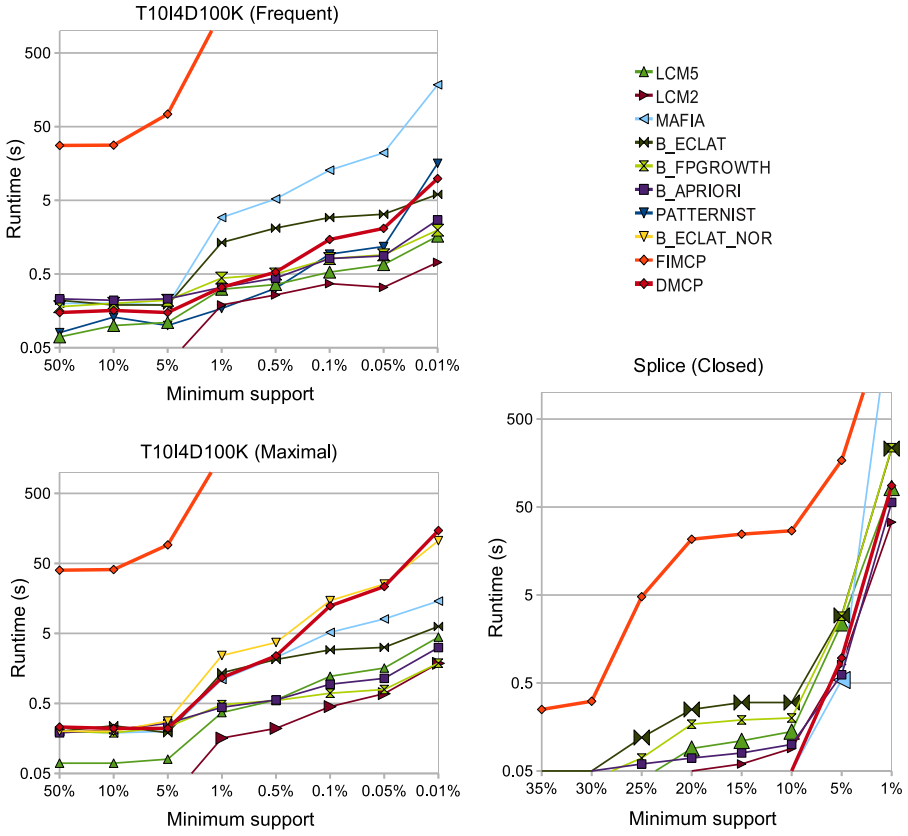


Figure 4.6: Run times of several algorithms on T10I4D100K and Splice.

insights allowing us to create an integrated CP solver based on the ideas of fine-grained constraint activation; it uses boolean vectors as basic type, supports general matrix constraints, enables multiple data representations and uses an auxiliary store inspired by the efficient constraint evaluations of itemset mining algorithms. Additionally, we demonstrated how the framework can be used for complexity analysis of mining tasks and illustrated this on a problem in graph mining. We showed experimentally that our system overcomes most performance differences.

Many questions have still been left unanswered. The general matrix constraint presented has a notation close to mathematical programming, but the applicability of such techniques was out of the scope of this work. We

focussed on the integration with constraint programming and at the moment, we implemented the optimized propagators in a new CP system which does not support the wide range of constraints general CP systems such as Gecode do. Whether it is possible to include the same optimizations in Gecode is an open question. Another question is which other itemset mining strategies can be incorporated in a general constraint programming setting. An essential component of our current approach is that constraints express a relationship between items and transactions. However, other itemset mining systems, of which FP-Growth [Han et al. 2000] is the most well-known example, represent the data more compactly by merging identical transactions during the search; as our current approach is built on fixed-length vectors, transactions cannot be merged in the same way.

Chapter 5

k-Pattern Set Mining using CP

This chapter[1] introduces the general problem of k-pattern set mining as well as a high-level language for expressing such problems. This language can be transformed into a CP specification, building on the constraints introduced in Chapter 3. We subsequently investigate the possibilities of exhaustive search for k-pattern set mining, while Chapter 6 investigates other search strategies.

5.1 Introduction

We introduce the problem of k-pattern set mining, concerned with finding a set of k related patterns under constraints. This contrasts to regular pattern mining where one searches for many individual patterns, such as in the previous chapters. The k-pattern set mining problem is a very general problem that can be instantiated to a wide variety of well-known mining tasks including concept-learning, rule-learning, redescription mining, conceptual clustering and tiling.

In this chapter, we take a first step towards a general purpose approach to pattern set mining. To this end, we introduce a general high-level modeling language, independent of underlying frameworks. We formulate a large number of constraints for use in k-pattern set mining, both at the local level, that is,

[1] Based on the journal paper "k-Pattern Set Mining under Constraints" [Guns et al. 2011c], also published as technical report [Guns et al. 2010a].

on individual patterns, and on the global level, that is, on the overall pattern set. Both levels of constraints are formalized at the same time, in a single specification. For many well-known tasks we show how they can be modeled in this high-level modeling language.

Our vision is that these declaratively specified problems are solved using a general solver in a uniform way. Developing such solvers is a challenge. In this chapter, we build on the constraint programming framework introduced in Chapter 3. We present a mapping of pattern set constraints to constraints currently available in CP. This allows us to investigate a large number of settings within a unified framework. It will also allow us to gain insight in the possibilities and limitations of using a CP solver for the challenging pattern set mining problems.

A potential issue is that standard CP solvers employ exhaustive search. When mapping a pattern set mining task specification to a CP specification, a CP solver will solve this exhaustively too. Solving pattern set mining problems exhaustively is uncommon in data mining, hence we cannot expect that this approach is feasible in all cases. We investigate this issue by considering the following questions:

- which pattern set mining tasks can be solved in reasonable time, using existing CP technology?
- which modeling choices can help to more effectively solve pattern set mining problems?

Our contribution to answering these questions is through the categorisation of constraints used in pattern set mining. Constraints are categorized by how effectively they can be propagated during the search process. This contributes new insights into the relation between local and global constraints. This is important as it allows us to create guidelines on how to model new problems successfully and how to model existing problems more efficiently.

Furthermore, we hope that such insights could also be used for developing more heuristic declarative approaches, for instance, when combining local search with exhaustive search. Such alternative search methods for pattern set mining are investigated in the next chapter.

5.2 k-Pattern Set Mining

One of the major problems of *local* pattern mining is that the set of solutions is typically too large. In Chapter 3 we studied how constraints can be used to decrease the number of patterns. However, this is often not enough and the patterns need to be post-processed to become useful. This resulted in the adoption of *two step* procedures in order to arrive at a small set of highly relevant patterns. First, all patterns that adhere to some chosen *local constraints* are mined exhaustively. Then, these patterns are combined under a set of *global constraints*, often including an optimisation function f. Because of the size of the local pattern set, in the second stage usually heuristic techniques are used when searching for the approximately best pattern set.

Here we wish to avoid this two step procedure by formulating both local and global constraints together, that is, on a pattern set that will contain k patterns where k is a fixed number. The problem of *k-pattern set mining under constraints* can be defined, similar to the general pattern set mining problem (Equation (2.3)), as that of finding:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \in \mathcal{L}^k \mid p(\Pi, \mathcal{D}) \text{ is true}\}. \tag{5.1}$$

The pattern set Π consists of k patterns π. The pattern set Π as a whole forms a global model. The constraint p specifies both local and global constraints at the overall pattern set level. In addition, as the number of pattern sets can become very large, we will study how to find the best pattern set with respect to an optimisation criterion $f(\Pi)$. That is, we study how to find the optimal pattern set Π by searching, for example, for the one with maximum $f(\Pi)$:

$$\arg \max_{\Pi \in \mathrm{Th}(\mathcal{L}, p, \mathcal{D})} f(\Pi),$$

where we still have the possibility to impose constraints through p.

In the rest of this chapter, we assume that a k-pattern set Π consists of k individual itemset patterns π. Every pattern π is represented by its itemset I and transaction set T: $\pi = (I, T)$. We write $\Pi = \left(\pi^1, \ldots, \pi^k\right) = \left((I^1, T^1), \ldots, (I^k, T^k)\right)$.

5.2.1 Families of Constraints

In this section we present four families of constraints. In the next section, we then show how k-pattern set mining problems can be specified as combinations of constraints from these families. The first family is the family of *individual*

pattern constraints. The typical local pattern mining constraints of Chapter 3 fall into this category. Second, *redundancy constraints* can be used to constrain or minimise the redundancy between different patterns. *Coverage constraints* deal with defining and measuring how well a pattern set covers the data. Given labelled data, the *discriminative constraints* can be used to measure and optimise how well a pattern or pattern set discriminates between positive and negative examples.

In this chapter we use the notation *predicate* $\Leftrightarrow$ *equation* to define a constraint predicate in terms of an equation consisting of operations on patterns (itemsets and transaction sets).

Individual Pattern Constraints

We quickly review the most important constraint-based mining constraints from Chapter 3 and introduce a predicate for each constraint.

Coverage constraint. The coverage constraint explicitly constrains the set of transactions in a pattern $\pi = (I, T)$ to those transactions containing the itemset:

$$coverage(\pi) \Leftrightarrow (T = \varphi(I) = \{t \in \mathcal{S} \mid \forall i \in I : (t, i) \in \mathcal{D}\}). \tag{5.2}$$

Disjunctive coverage constraint. Alternatively, we can define that a transaction is covered if it is covered by at least one item. This is useful when mining for a disjunction of items.

$$disjcoverage(\pi) \Leftrightarrow T = \{t \in \mathcal{S} \mid \exists i \in I : (t, i) \in \mathcal{D}\}. \tag{5.3}$$

Closedness constraint. Removes redundant solutions among the patterns found, when used together with the coverage constraint.

$$closed(\pi) \Leftrightarrow (I = \psi(T) = \{i \in \mathcal{I} \mid \forall t \in T : (t, i) \in \mathcal{D}\}). \tag{5.4}$$

Size constraints. The size of a pattern $\pi = (I, T)$ can straightforwardly be measured as $size(\pi) = |I|$. In the general case, we can define a lower or upper bound constraint on any measure. With $\lesseqgtr$ we will denote any comparison $\lesseqgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$. A constraint on the size of the pattern can thus be formulated as

$$size(\pi) \lesseqgtr \theta \Leftrightarrow |I| \lesseqgtr \theta. \tag{5.5}$$

Frequency constraint. The frequency of an itemset is simply the size of its transaction set: $freq(\pi) = |T|$, which can also be constrained as $|T| \lessgtr \theta$.

$$freq(\pi) \lessgtr \theta \Leftrightarrow |T| \lessgtr \theta. \tag{5.6}$$

Redundancy Constraints

An important problem in local pattern mining is that of redundancy among patterns. We already defined the *closedness* constraint that can be used to remove a certain kind of redundancy of individual patterns. However a pattern in a set can still be considered logically redundant if it covers *approximately* the same transactions as another pattern in the set. Alternatively, one can consider patterns to be redundant if their syntax is similar, for example, if two itemsets share many of the same items. In the following, we will ignore syntactic similarity and will instead measure the redundancy between two patterns by the the similarity or distance between their transaction sets.

Distance measures. We can measure the overlap between two patterns as the size of the intersection between the transaction sets. Likewise, the distinctness of two patterns can be measured by the size of the symmetric difference between the transaction sets.

$$overlap(\pi^1, \pi^2) = \left|T^1 \cap T^2\right|, \tag{5.7}$$

$$distinct(\pi^1, \pi^2) = \left|(T^1 \cup T^2) \setminus (T^1 \cap T^2)\right|. \tag{5.8}$$

The distance between two patterns can also be measured using any distance measure between the two transaction sets, for example the Jaccard similarity coefficient or the Dice coefficient.

$$jaccard(\pi^1, \pi^2) = \frac{|T^1 \cap T^2|}{|T^1 \cup T^2|}, \tag{5.9}$$

$$dice(\pi^1, \pi^2) = \frac{2 * |T^1 \cap T^2|}{|T^1| + |T^2|}. \tag{5.10}$$

Such measures can be constrained by a comparison operator $\lessgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$ and a threshold θ. The measures above are defined on the transaction sets T, but could equivalently be calculated on the itemsets I.

Combining measures. Since the measures only indicate the redundancy between two patterns, and not an entire pattern set, we often need to aggregate over all pairwise combinations of patterns. A first approach is to constrain the sum

of all pairwise evaluations. Using the function name *dist()* to indicate any distance measure, we may define that:

$$sumdist(\Pi) \lesseqgtr \theta \Leftrightarrow \sum_{a<b} dist(\pi^a, \pi^b) \lesseqgtr \theta. \tag{5.11}$$

An alternative approach is to bound the minimum or maximum value over all evaluations:

$$mindist(\Pi) \lesseqgtr \theta \Leftrightarrow \min_{a<b} \big(dist(\pi^a, \pi^b)\big) \lesseqgtr \theta. \tag{5.12}$$

Both approaches will constrain the distance between all patterns in the pattern set, and hence constrain the redundancy. However, depending on the formulation, the constraints will be easier or harder to evaluate and propagate during search.

For example, in the case $mindist \geq \theta$, we can constrain every pairwise evaluation separately and independently:

$$\begin{aligned} mindist(\Pi) \geq \theta &\Leftrightarrow \min_{a<b} \big(dist(\pi^a, \pi^b)\big) \lesseqgtr \theta \\ &\Leftrightarrow \forall_{a<b}\ dist(\pi^a, \pi^b) \lesseqgtr \theta. \end{aligned}$$

Coverage Constraints

The cover of a single pattern is simply $cover(\pi) = T$, because of the above defined *coverage constraint*. The entire pattern set also covers transactions.

Pattern set cover. A pattern set can be interpreted as a disjunction of the individual patterns. Hence, we calculate the transaction set of the entire pattern set by taking the union over the individual transaction sets:

$$cover(\Pi) = T^{\Pi} = T^1 \cup \cdots \cup T^k. \tag{5.13}$$

Frequency of pattern set. The frequency of the pattern set is then calculated exactly like the frequency of an individual pattern: $freq(\Pi) = |T^{\Pi}|$. This can again be constrained:

$$freq(\Pi) \lesseqgtr \theta \Leftrightarrow |T^{\Pi}| \lesseqgtr \theta. \tag{5.14}$$

Area of pattern set. The area of a pattern set was studied in the context of large tile mining [Geerts et al. 2004]. The tile of a pattern contains all tuples $(t, i) \in \mathcal{D}$ that are covered by the pattern: $tile(\pi) = \{(t, i) \,|\, t \in T, i \in I\}$.

These tuples form a tile or rectangle of 1's in the binary database D. The area of a single pattern is the number of tuples that are covered in the tile: $area(\pi) = |tile(\pi)| = |I| \cdot |T|$. The area of a pattern set can now be defined as the union of all the tiles of the individual patterns. Note that tiles can be overlapping, but every tuple (t, i) covered is only counted once.

$$area(\Pi) = \left|tile(\pi^1) \cup \ldots \cup tile(\pi^k)\right|. \tag{5.15}$$

Discriminative Constraints

Discriminative constraints measure and constrain how well a pattern can discriminate datasets from one another. We consider the case where two databases are given, $\mathcal{D}^+$ and $\mathcal{D}^-$. The positive cover of a pattern is the cover of the pattern on the positive examples: $cover^+(\pi) = T \cap \mathcal{S}^+$. Similarly, the negative cover is $cover^-(\pi) = T \cap \mathcal{S}^-$.

In a similar way, the closedness constraint can be defined only on the positive or negative transactions: $closed^+(\pi) \Leftrightarrow I = \psi(T \cap \mathcal{S}^+)$ respectively $closed^-(\pi) \Leftrightarrow I = \psi(T \cap \mathcal{S}^-)$.

The positive and negative cover of the entire pattern set can be calculated similar to the positive/negative cover of a single pattern, where T^Π is defined as in equation (5.13):

$$cover^+(\Pi) = T^\Pi \cap \mathcal{S}^+, \tag{5.16}$$

$$cover^-(\Pi) = T^\Pi \cap \mathcal{S}^-. \tag{5.17}$$

The number of positive/negative examples covered by an entire pattern set is then calculated as:

$$freq^+(\Pi) = |cover^+(\Pi)| = |T^\Pi \cap \mathcal{S}^+|, \tag{5.18}$$

$$freq^-(\Pi) = |cover^-(\Pi)| = |T^\Pi \cap \mathcal{S}^-|. \tag{5.19}$$

Accuracy of a pattern set. Accuracy is defined as the proportion of examples that are correctly covered by the pattern set. Let p and n be the number of positive and negative examples covered, and P and N the total number of positives and negatives. Accuracy is then calculated by $(p + (N - n))/(P + N)$. Substituting p and n by equations (5.18) and (5.19), we get the following formulation:

$$accuracy\left(freq^+(\Pi), freq^-(\Pi)\right) = \frac{freq^+(\Pi) + |\mathcal{S}^-| - freq^-(\Pi)}{|\mathcal{S}|}. \tag{5.20}$$

Note that when maximizing the accuracy of a pattern set, the constants $|\mathcal{S}|$ and $|\mathcal{S}^-|$ can be removed and one can simply optimize the sum $freq^+(\Pi) - freq^-(\Pi)$ [Fürnkranz and Flach 2005].

Using the same principles as above, we can also define other discriminative measures such as relative accuracy and the Laplace estimate:

$$rel\_accuracy : \frac{freq^+(\Pi)}{|\mathcal{S}^+|} - \frac{freq^-(\Pi)}{|\mathcal{S}^-|}, \tag{5.21}$$

$$Laplace : \frac{freq^+(\Pi) + 1}{freq^+(\Pi) + freq^-(\Pi) + 2}. \tag{5.22}$$

The topic of identifying such measures has already been studied extensively in pattern mining [Morishita and Sese 2000; Liu et al. 1998] and in rule learning [Fürnkranz and Flach 2005]. We studied the use of discriminative constraints for local pattern mining in Chapter 3.4.

5.2.2 Instantiations

As argued in the introduction, the k-pattern set mining problem is very general and flexible. One of the contributions of this chapter is that we show how it can be instantiated to address several well-known data mining problems. This is explained in the following paragraphs. We will present both satisfaction problems and optimisation problems.

Satisfaction problems are specified by listing all the constraints needing to be satisfied. Optimisation problems additionally start with the maximise or minimise keywords, indicating the function to optimise. A solution needs to satisfy all constraints; in case of an optimisation problem, only the solution with the highest, or lowest, value for the given optimisation function is sought.

$k-$term DNF Learning and Concept Learning

The main aim when learning a $k-$term formula in disjunctive normal form (DNF) is to learn a formula which performs a binary prediction task as accurately as possible, given a set of labelled training examples. A formal definition of $k-$term DNF learning was given in [Kearns and Vazirani 1994]. Within our framework, we can interpret each clause as an itemset, while the pattern set corresponds to a disjunction of such clauses. We can formalise this

problem as finding pattern sets Π satisfying:

$$\begin{aligned} \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}^{+}(\pi), \\ & \mathit{accuracy}\left(\mathit{freq}^{+}(\Pi), \mathit{freq}^{-}(\Pi)\right) \geq \theta. \end{aligned} \tag{5.23}$$

where we choose $\theta = |\mathcal{S}^{+}|$ if we do not wish to allow for errors on the training data (pure DNF learning). Note that we have added a closedness constraint on the positive transactions; the main motivation for this choice is that for any $k-$term DNF containing arbitrary itemsets, there is an equally good or better $k-$term DNF with only closed-on-the-positive itemsets [Garriga et al. 2008]. Adding this constraint also reduces the space of hypotheses and hence reduces the practical complexity of the problem.

The above formulation would result in all k-pattern sets that are accurate enough. In the concept learning setting we are usually interested only in discovering the most accurate pattern set. To achieve this, the above satisfaction problem is turned into an optimisation problem:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{accuracy}\left(\mathit{freq}^{+}(\Pi), \mathit{freq}^{-}(\Pi)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}^{+}(\pi). \end{aligned}$$

We can replace accuracy with any other discriminative constraint as explained in Section 5.2.1. Note that it is easy to add other constraints to this formulation, like a minimum frequency constraint on every individual pattern:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{accuracy}\left(\mathit{freq}^{+}(\Pi), \mathit{freq}^{-}(\Pi)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}^{+}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{freq}(\pi) \geq \theta. \end{aligned}$$

In all cases, the result will be a k-pattern set with high accuracy on the examples. Every pattern π will represent a learned concept.

Conceptual Clustering

The main aim of clustering algorithms is to find groups of examples which are similar to each other. In conceptual clustering, the additional goal is to

learn a conceptual description for each of these clusters [Fisher 1987]. In this section, we consider a simplified version of conceptual clustering, in which we call two examples similar if they contain the same pattern. Hence, each cluster is described by a pattern, and all examples that are covered by the pattern are part of the cluster. We then formalise conceptual clustering as finding pattern sets Π that do not overlap and cover all the examples:

$$\begin{aligned} \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}(\pi), \\ & \mathit{cover}(\Pi) = \mathcal{S}, \\ \forall \pi^a, \pi^b \in \Pi: \quad & \mathit{overlap}(\pi^a, \pi^b) = 0. \end{aligned}$$

The solutions to this model form the restricted set of clusterings that do not overlap. Still, finding all such clusterings may not be desirable and finding one clustering could be sufficient. In this case, it could be more interesting to search for the best non-overlapping clustering. There are multiple ways to define what the 'best' clustering is. One possible way is to prefer solutions in which the sizes of the clusters do not differ too much from each other. We can formalize this in several possible ways. For example, we could search for solutions in which the minimum size of the clusters is as large as possible:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \min\left(\mathit{freq}(\pi^1), \ldots, \mathit{freq}(\pi^k)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}(\pi), \\ & \mathit{cover}(\Pi) = \mathcal{S}, \\ \forall \pi^a, \pi^b \in \Pi: \quad & \mathit{overlap}(\pi^a, \pi^b) = 0. \end{aligned}$$

The minimum cluster size would be maximal if all clusters have the same size; hence this formulation will prefer more balanced solutions. Alternatively, one could also use the following optimization criterion:

$$\underset{\Pi}{\text{minimise}} \quad \max\left(\mathit{freq}(\pi^1), \ldots\right) - \min\left(\mathit{freq}(\pi^1), \ldots\right);$$

this would enforce a small difference between cluster sizes more directly. We will compare these two settings in the experimental section.

In the general case, other settings may also be desirable. For instance, the constraint that clusters are not allowed to overlap might seem too restrictive, and one could choose to use the following optimisation criterion:

$$\underset{\Pi}{\text{minimise}} \sum_{\pi^a, \pi^b \in \Pi} \mathit{overlap}(\pi^a, \pi^b),$$

however, we determined experimentally that in many datasets a set of k non-overlapping patterns can be found, and hence this optimization criterion would give the same solution as when using the overlap constraint.

Further extensions of the model are needed in order to make the setting more useful. Of most interest is probably the incorporation of arbitrary distance functions in the optimisation, as is common in clustering. We here restrict ourselves to measures such as *overlap* that can be defined over the item and transaction sets of the patterns.

$k-$Tiling

The main aim of tiling [Geerts et al. 2004] is to cover as many 1s in a binary matrix as possible, with a given number of patterns or tiles. A tiling can be considered useful as the patterns in a tiling are in some way most characteristic for the data. We can formalise this problem as follows:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{area}(\Pi), \\ \forall \pi \in \Pi : \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi : \quad & \mathit{closed}(\pi). \end{aligned}$$

The closedness constraint is not strictly necessary, but closed itemsets cover a larger area than their non-closed counterparts.

Redescription Mining

The main aim of redescription mining [Parida and Ramakrishnan 2005] is to find sets of syntactically different formulas that all cover the same set of transactions; such sets of formulas are of interest as they point towards equivalences in the attributes in the data. We assume given a number of disjoint partitions of items $\mathcal{I}^1 \ldots \mathcal{I}^k$ where $\forall p : \mathcal{I}^p \subseteq \mathcal{I}$ and $\forall p, q, p \neq q : \mathcal{I}^p \cap \mathcal{I}^q = \emptyset$ and can formalize the problem in multiple alternative ways.

We restrict ourselves to finding conjunctive formulas that form redescriptions. In this setting, we may search for a pattern set of size k, where k is the number of partitions. Every pattern in $\pi^1 \ldots \pi^k$ will be a subset of a partition in $\{\mathcal{I}^1, \ldots, \mathcal{I}^k\}$. When searching for exact redescriptions, the cover of each pattern will have to be identical. We can now formalize the redescription

mining problem of finding the largest exact redescription, as follows:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{freq}(\pi^1), \\ \forall \pi^a \in \Pi: \quad & I^a \subseteq \mathcal{I}^a \\ \forall \pi \in \Pi: \quad & \mathit{covered}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}(\pi), \\ \forall \pi^a, \pi^b \in \Pi: \quad & T^a = T^b, \end{aligned}$$

Note that maximizing the frequency of the first pattern is sufficient for this task as the cover, and hence the frequency, of each pattern is exactly the same.

As this may be a too stringent requirement, one could also search for approximate redescriptions that cover a sufficient number of examples:

$$\begin{aligned} \underset{\Pi}{\text{minimise}} \quad & \mathit{sumdist}(\Pi), \\ \forall \pi \in \Pi: \quad & I \subseteq \mathcal{I}^\pi \\ \forall \pi \in \Pi: \quad & \mathit{covered}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{freq}(\pi) \geq \theta, \end{aligned}$$

The $\mathit{sumdist}(\Pi)$ measures the distance between the different transaction sets of the patterns. The redescription that minimizes this measure will cover as much as possible the same transactions. In principle every distance measure from Section 5.2.1 can be used.

We compare these two settings further in the experimental section.

5.3 Constraint Programming Model

With its declarative modeling language and constraint-based search, constraint programming contains all the necessary elements to address the general problem of k-pattern set mining. Two questions remain; how can k-pattern set mining problems be specified in a CP system, and how effective will the search be?

In local pattern mining, the search space of itemsets has size $O(2^n)$, where $n = |\mathcal{I}|$, the number of items in the database. Clearly, a naïve algorithm that would simply enumerate and test each pattern would not perform well. Thanks to the effective propagation of the constraints involved, for example the well-known frequency constraint, fast and efficient algorithms exist that do

exhaustive search without having to enumerate each pattern. For k-pattern set mining, when searching for a set of k patterns directly from the data in one step, the search space has size $O(2^{nk})$ with n the number of items in the database and k the number of patterns in the set. Hence, to do exhaustive search, it becomes even more crucial to have constraints that effectively prune the search space.

In this section, we first study how the constraints of section 5.2.1 can be modeled in a constraint programming framework, and how effective their propagation will be. We then take a closer look at how the problem instantiations are modeled in the framework.

5.3.1 Modeling Constraints

Constraints in pattern mining are typically divided into two broad categories: local constraints and global constraints. A constraint is local when it is defined on one individual pattern, and global when it is defined on multiple patterns. This definition of a global constraint differs from the usual definition in constraint programming, where a global constraint indicates a constraint that relates a number of variables in a non-trivial way. During the study of the effectiveness of each constraint, we identify two new and special categories. Within the category of local constraints we identify local look-ahead constraints, and within the category of global constraints, the pairwise global constraints. This further distinction will give us a better understanding of the effectiveness of the constraints, which will help us to better understand the search behaviour of the models at large.

In our study all patterns are itemsets. As in Chapter 3 we will represent a pattern's tuple $\pi = (I, T)$ by introducing a boolean variable for every item i and every transaction identifier t. A pattern set of size k simply consists of k such patterns: $\forall_{p=1..k} : \pi^p = (I^p, T^p)$.

Individual Pattern Constraints

We have extensively studied constraints on individual patterns in Chapter 3. Table 5.1 provides an overview of the most common constraints, their set notation and how they can be expressed in a constraint programming framework.

We also categorize the constraints based on their propagation power. Individual pattern constraints are by definition local constraints. However, we make a further distinction for constraints that can be expressed in reified form. We

constraint category constraint name	set notation	CP
local look-ahead		
$coverage(\pi)$	$T = \varphi(I)$	$\forall t \in \mathcal{S} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0$
$disjcoverage(\pi)$	$T = \alpha(I)$	$\forall t \in \mathcal{S} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} > 0$
$closed(\pi)$	$I = \psi(T)$	$\forall i \in \mathcal{I} : I_i \leftrightarrow \sum_{t \in \mathcal{S}} T_t(1 - \mathcal{D}_{ti}) = 0$
$freq(\pi) \geq \theta$	$\lvert T \rvert \geq \theta$	$\forall i \in \mathcal{I} : I_i \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta$
$size(\pi) \geq \theta$	$\lvert I \rvert \geq \theta$	$\forall t \in \mathcal{S} : T_t \rightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} \geq \theta$
regular local		
$freq(\pi) \leq \theta$	$\lvert T \rvert \leq \theta$	$\sum_{t \in \mathcal{S}} T_t \leq \theta$
$size(\pi) \leq \theta$	$\lvert I \rvert \leq \theta$	$\sum_{i \in \mathcal{I}} I_i \leq \theta$

Table 5.1: Individual pattern constraints

call such constraints *local look-ahead* constraints because they can 'look ahead' for every reified variable and determine whether it can no longer contribute to a valid solution. The resulting difference in propagation power motivates us to categorize local look-ahead constraints as a special kind of local constraint.

Redundancy Constraints

In Table 5.2 we give an overview of the redundancy constraints and how to combine them, as explained in Section 5.2.1.

Distance measures. In CP the cardinality of a set can in principle be calculated by summing 0/1 variables representing the elements in the set. However, to deal with redundancy, we often need to calculate the cardinality of a set which is the result of comparing two other sets. Representing the intermediary set with additional variables would make our notation cumbersome. For instance, to calculate the overlap $overlap(\pi^1, \pi^2) = |T_1 \cap T_2|$, we would first need to calculate the set $T_1 \cap T_2$, and then sum the variables representing this set. As a short-hand notation we will therefore combine the set operation and the size

constraint category constraint name	set notation	CP
pairwise		
overlap(π^1, π^2)	$\lvert T^1 \cap T^2 \rvert$	$\sum_{t \in \mathcal{S}} [T_t^1 + T_t^2 = 2]$
distinct(π^1, π^2)	$\lvert (T^1 \cup T^2) \setminus (T^1 \cap T^2) \rvert$	$\sum_{t \in \mathcal{S}} [T_t^1 + T_t^2 = 1]$
jaccard(π^1, π^2)	$\frac{\lvert T^1 \cap T^2 \rvert}{\lvert T^1 \cup T^2 \rvert}$	$\frac{\sum_{t \in \mathcal{S}} [T_t^1 + T_t^2 = 2]}{\sum_{t \in \mathcal{S}} [T_t^1 + T_t^2 \geq 1]}$
$\min_{a<b}$ *dist*$(\pi^a, \pi^b) \geq \theta$	$\min_{a<b}$ *dist*$(\pi^a, \pi^b) \geq \theta$	$\forall a < b$: *dist*$(\pi^a, \pi^b) \geq \theta$
global		
$\sum_{a<b}$ *dist*$(\pi^a, \pi^b) \lesseqgtr \theta$	$\sum_{a<b}$ *dist*$(\pi^a, \pi^b) \lesseqgtr \theta$	$\sum_{a<b} V_{ab} \lesseqgtr \theta$, $V_{ab} =$ *dist*(π^a, π^b)
$\min_{a<b}$ *dist*$(\pi^a, \pi^b) \leq \theta$	$\min_{a<b}$ *dist*$(\pi^a, \pi^b) \leq \theta$	$\min_{a<b} V_{ab} \leq \theta$, $V_{ab} =$ *dist*(π^a, π^b)

Table 5.2: Redundancy constraints and their combinations

calculation as follows:

$$|T^1 \cap T^2| = \sum_{t \in \mathcal{S}} [T_t^1 + T_t^2 = 2], \tag{5.24}$$

where we count the number of transactions for which $T_t^1 + T_t^2 = 2$ is true by using the Iverson bracket notation $[\cdot]$. Redundancy constraints measure the distance between two patterns, so they are by nature pairwise constraints.

Combining measures. The goal is not to constrain one pair of patterns, but all the pairs in a pattern set. The way in which this aggregation is done defines the complexity of constraining the redundancy of the overall pattern set. Constraining all the distances at once results in one large global constraint. An example of this is when we sum over all the pairwise distances:

$$\sum_{a<b} dist(\pi^a, \pi^b) \lesseqgtr \theta. \tag{5.25}$$

This constraint would not propagate very well as a change in one pattern does not directly influence the other patterns. Typically many of the distances would have to be known before the constraint can propagate a change on the other distances. More effective is to constrain every distance individually; consider

constraint category constraint name	set notation	CP
global		
$freq(\Pi) \lesseqgtr \theta$	$\lvert\bigcup_\pi T\rvert \lesseqgtr \theta$	$\sum_{t\in\mathcal{S}} B_t \lesseqgtr \theta,$ where $B_t = \left[\left(\sum_{p\in\{1..k\}} T_t^p\right) \geq 1\right]$
$accuracy(\Pi) \lesseqgtr \theta$	$\left(freq^+(\Pi) - freq^-(\Pi)\right) \lesseqgtr \theta$	$\left(\sum_{t\in\mathcal{S}^+} B_t - \sum_{t\in\mathcal{S}^-} B_t\right) \lesseqgtr \theta,$ where $B_t = \left[\left(\sum_{p\in\{1..k\}} T_t^p\right) \geq 1\right]$
$area(\Pi) \lesseqgtr \theta$	$\lvert\bigcup_\pi (I \times T)\rvert \lesseqgtr \theta$	$\sum_{i\in\mathcal{I},t\in\mathcal{S}} B_{it} \lesseqgtr \theta,$ where $B_{it} = \left[\left(\sum_{p\in\{1..k\}} [I_i^p + T_t^p = 2]\right) \geq 1\right]$

Table 5.3: Coverage and discriminative constraints

the case where we want to constrain the smallest pairwise distance to be larger than a certain threshold θ:

$$\left(\min_{a<b} dist(\pi^a, \pi^b)\right) \geq \theta. \tag{5.26}$$

In this case, we can put a constraint on each pair separately: $\forall a < b : dist(\pi^a, \pi^b) \geq \theta$. In this case, we can decompose the global constraint in a number of independent pairwise constraints. This difference in propagation strength motivates us to discriminate pairwise constraints from regular global constraints.

Coverage and Discriminative Constraints

In Table 5.3 we list the coverage and discriminative constraints previously discussed. The cover of the entire pattern set depends on the cover of every individual pattern $T^\Pi = T^1 \cup \cdots \cup T^k$. If a transaction is covered by one pattern, it is covered by the pattern set. Hence, a transaction t is covered iff

$\exists \pi \in \Pi : T_t = 1$. In constraint programming we can model this by introducing a temporary variable B_t for every transaction t, where $B_t \leftrightarrow (\sum_{p \in \{1 \ldots k\}} T_t^p) \geq 1$. Coverage and discriminative constraints, which we originally defined on the transaction sets of individual patterns, can also be defined on such temporary variables. Because of the indirect relation between patterns through these temporary variables, coverage and discriminative constraints are categorised as regular global constraints.

A special case is the *area* constraint, for which we need to calculate the number of ones in the matrix that is covered by the set of patterns. We can model this by introducing a temporary variable B_{it} for every element in the matrix: $B_{it} = 1$ iff at least one tile covers that element, that is, $\exists \pi \in \Pi : (I_i = 1 \wedge T_t = 1)$. Equivalently $B_{it} = \left[\left(\sum_{p \in \{1..k\}} [I_i^p + T_t^p = 2]\right) \geq 1\right]$. This constraint cannot propagate the truth-value of B_{it} well: if $B_{it} = 1$ then it should propagate that at least 1 pattern covers this element, but this is only possible if all but one pattern are known not to cover it. Additionally, the area constraint contains $|\mathcal{I}| * |\mathcal{S}|$ such variables, where $|\mathcal{S}|$ is typically large. Hence, we can expect this constraint to have particularly weak propagation.

Symmetry Breaking Constraints

We model the set of k patterns in constraint programming by means of an array of k patterns. An artefact of this is that syntactic variations of the same pattern set can be generated, for example (π_1, π_2) and (π_2, π_1). This is something well-known in constraint programming that can be solved by means of so-called *symmetry breaking constraints* [Rossi et al. 2006]. In many cases, symmetry breaking constraints impose a strict ordering on the patterns in the pattern set.

There are many ways to impose an ordering on patterns. The most straightforward way is to impose a simple lexicographic ordering:

$$symm\_breaking(\Pi) : \pi^1 < \pi^2 < \ldots < \pi^k \tag{5.27}$$

Given that the order constraint can be enforced between every pair of patterns, this constraint falls in the category of *pairwise constraints*.

There are some design decisions to be made when ordering patterns. Evidently, one can impose a lexicographic ordering on the itemsets of the patterns in the array. In case every pattern is a closed itemset, the itemset uniquely defines the transaction set and the transaction set uniquely defines the itemset. Hence instead of lexicographically ordering the itemsets, one can also order the

transaction sets. As the transaction set is typically larger than the itemset, this could potentially lead to better propagation as more variables are involved.

Alternatively, one could order the patterns primarily on a property like frequency or accuracy, and additionally use a lexicographic order on the item- or transaction sets to break ties. Finally there is also the choice to post the order constraints only on subsequent patterns, or between all pairs of patterns. Posting them between all pairs requires $(n-1)(n-2)/2$ additional constraints, but could result in additional pruning.

5.3.2 Modeling Instantiations

The constraints introduced in the previous section allow us to model all the mining problems that were introduced in Section 5.2.2. Essentially, to obtain a complete CP model, we need to enter the appropriate CP formulations of constraints in the problem descriptions of Section 5.2.2. Let us illustrate this for the problem of concept learning.

One step When we fill in the formulas of the previous section in the model of equation (5.23), we obtain this CP model:

$$\forall p \in \{1,\ldots,k\} : \forall t \in \mathcal{S} : T_t^p \leftrightarrow \sum_{i \in \mathcal{I}} I_i^p (1 - \mathcal{D}_{ti}) = 0, \qquad (Coverage)$$

$$\forall p \in \{1,\ldots,k\} : \forall i \in \mathcal{I} : I_i^p \leftrightarrow \sum_{t \in \mathcal{S}^+} T_t^p (1 - \mathcal{D}_{ti}) = 0, \qquad (Closed^+)$$

$$\forall t \in \mathcal{S} : B_t = \Big[\big(\sum_{p \in \{1..k\}} T_t^p \big) \geq 1 \Big], \qquad (Cover\ of\ set)$$

$$accuracy\big(\sum_{t \in \mathcal{S}^+} B_t, \sum_{t \in \mathcal{S}^-} B_t \big) \geq \theta, \qquad (Accurate)$$

$$T^1 < T^2 < \ldots < T^k. \qquad (Symm.\ Br.)$$

This model captures a problem that involves k patterns at the same time; constraint programming systems provide a strategy for finding optimal solutions to this problem. The important advantage of this model is that it allows a one step solution to the concept learning problem.

Two step Nevertheless, one can also use CP systems to find an optimal solution in two steps. Closed-on-the-positive itemsets are also sufficient in

this case. First, all itemsets that fulfil the local constraints are mined:

$$\forall t \in \mathcal{S} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0, \qquad (Coverage)$$

$$\forall i \in \mathcal{I} : I_i \leftrightarrow \sum_{t \in \mathcal{S}^+} T_t(1 - \mathcal{D}_{ti}) = 0. \qquad (Closed^+)$$

Using all itemsets found, a new itemset database is created in which every item represents an itemset. On this new transactional database another pattern mining problem is solved, with the following constraints:

$$\forall t \in \mathcal{S} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} > 0, \qquad (Disj.\ coverage)$$

$$\sum_{i \in \mathcal{I}} I_i = k, \qquad (Size\ of\ set)$$

$$accuracy(\sum_{t \in \mathcal{S}^+} T_t, \sum_{t \in \mathcal{S}^-} T_t) \geq \theta. \qquad (Accurate)$$

Each resulting itemset corresponds to a set of local patterns and is hence a pattern set. The constraints enforce the desired size k and accuracy θ. This approach is similar to the approach chosen in [De Raedt and Zimmermann 2007], where additionally a frequency constraint is used. Our hope is that the single-step approach outperforms this more traditional step-wise approach. Whether this is the case will be explored in the next section.

5.4 Experiments

As shown in the previous section, constraint programming offers a powerful framework that fits the general problem of k-pattern set mining. Constraint programming has a generic search strategy of propagation and exhaustive search. The big question that we will try to answer in this section is: *How suitable is a generic CP approach, which uses exhaustive search, for pattern set mining?*

We have argued for the importance of constraints that propagate well, as this is needed to keep the search space manageable. In this section, we perform an experimental study on each of the introduced pattern set mining tasks. For each task, we will study the performance of the CP approach, and link these results to the constraints involved in the model. The aim of these experiments is to gain a better understanding in the suitability of CP systems for a broad range of pattern set mining tasks.

	Transactions	Items	Density	Class distr.
anneal	812	53	42%	77%
audiology	216	148	45%	26%
hepatitis	137	44	50%	81%
lymph	148	60	38%	55%
primary-tumor	336	31	48%	24%
soybean	630	50	32%	15%
tic-tac-toe	958	27	33%	65%
vote	435	48	33%	61%
zoo	101	36	44%	40%

Table 5.4: Characteristics of the datasets.

We focus our study on the following questions:

Q1 how does the proposed one step approach compare to the two step procedure?

Q2 how does the k-pattern set mining approach scale to the different tasks?

Q3 what is the relation between the performance of a model and the constraints involved?

We study these questions by performing experiments for each of the four tasks of interest, that is, concept-learning, clustering, tiling and rediscription mining. More experiments on the concept learning task are performed in the next chapter. We conclude this section with a discussion on the framework.

For the experiments we used the Gecode constraint programming solver [Gecode Team 2010]. It includes propagators for all the constraints used in this chapter. For constraints of the form $\sum[B^1 + B^2 \leqslant \alpha]$ we added a simple propagator that directly calculates the sum at large, thus avoiding to store an auxiliary variable for every single reified sum. The datasets used are from the UCI Machine Learning repository [Frank and Asuncion 2010] and were discretised using binary splits into eight equal-frequency bins. The majority class was chosen as the positive class. The properties of the resulting datasets are listed in Table 5.4. Experiments were performed on PCs running Ubuntu 8.04 with Intel(R) Core(TM)2 Quad CPU Q9550 processors and 4GB of RAM.

5.4.1 $k-$term DNF Learning and Concept Learning

This section focusses on the concept learning setting, where we want to maximise the accuracy of the pattern set. In case a k-pattern set is found

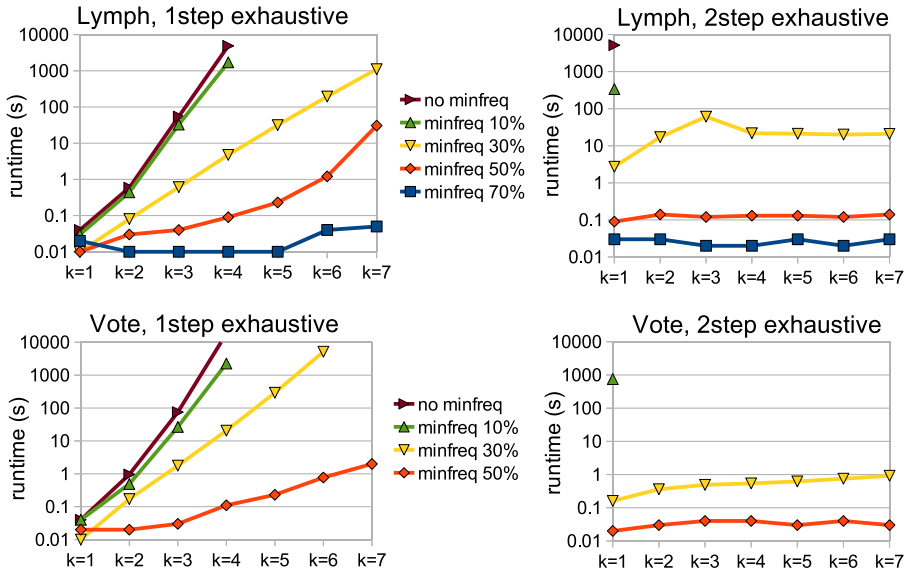


Figure 5.1: Comparing a one step exhaustive search with a two step exhaustive search to concept learning.

that covers all positive transactions and none of the negative ones, this setting is identical to pure k-term DNF learning.

We start by investigating the first question: how does the proposed one step approach compare to the two step procedure (Q1)? We compare both in our CP framework. In the two step approach, detailed in Section 5.3.2, first all patterns given a minimum frequency threshold are mined, and in the second step the best combination of k patterns is sought using constraint programming. In the one step approach we search for the k pattern set directly; a minimum frequency constraint is not needed, but we add it for comparison. Figure 5.1 shows the result of this experiment for the lymph and vote datasets. The two step approach performs very good for high frequency thresholds such as 50% and 70%; for low thresholds the two step approach can not handle the amount of candidate patterns that is generated in the first step. In our experiments, it could only handle up to a few thousand generated patterns. The one step approach on the other hand does not rely on the frequency constraint; even when there are potentially large amounts of local patterns, and hence the two-step method fails, the one step method can use the global constraints, up to certain values of k, to reduce the search space to a manageable size.

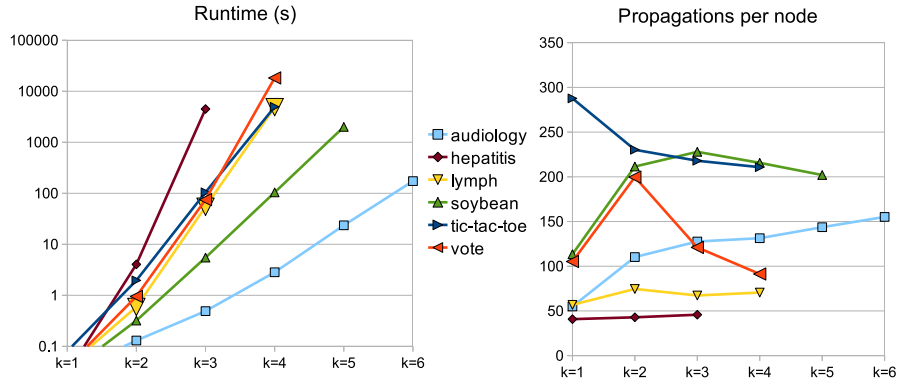


Figure 5.2: Runtime and number of propagations per node for the concept learning setting.

Next, we investigate the question how the approach scales on different datasets. In Figure 5.2 we study the k-pattern set mining problem of concept learning without a minimum frequency threshold on different datasets. For $k = 1$ we are able to find the single concept that best describes the data in less than a second. For $k = 2$ the two best concepts are also found in a reasonable amount of time, but for larger k the run times quickly become very large and the scalability is limited.

This can be explained by looking at the types of constraints involved, as posted in our third question (Q3). Overall, the concept learning model consists of the coverage and closed local look-ahead constraints which we already found to perform well in Chapter 3, as well as the global accuracy constraint and pairwise lexicographic ordering constraints (no frequency constraint is needed). The case $k = 1$ is special, as there are no ordering constraints and the accuracy constraint is local. For $k = 2$, there is a pairwise ordering constraint; the accuracy constraint is also pairwise as it is expressed on only two patterns. Starting from $k = 3$ more pairwise constraints are added, and the accuracy constraint becomes a regular global constraint.

Our hypothesis is that the more patterns are included in the pattern set, the less efficient the propagation of the global constraint becomes and hence the longer the search will take. To test this hypothesis, we plot the average number of constraint propagations per node of the search tree on the right of Figure 5.2. We observe that the number of propagations per node decreases or stays the same, except for the audiology dataset for which it increases moderately. Knowing that the number of nodes in the search tree grows quickly

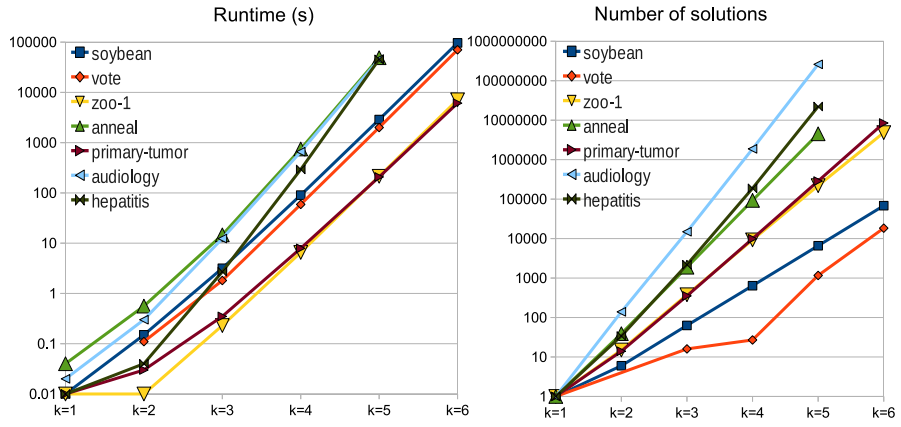


Figure 5.3: Runtime and number of conceptual clusterings for varying k.

for increasing k, a lack of increase in propagation means that many of those nodes will have to be visited. Hence for large k, the global accuracy constraint and the pairwise ordering do not allow for sufficient propagation to make the search feasible.

5.4.2 Conceptual clustering

In conceptual clustering, one is interested in finding clusters that are described by conceptual descriptions. In our case the conceptual description is an itemset, and the corresponding transactions constitute the cluster. The goal is to find a clustering with a certain number k of non-overlapping clusters.

To address the question how CP scales to different tasks (Q2), we first consider the differences between the constraint satisfaction setting (in which we wish to find all solutions that satisfy the constraints), and optimisation settings, as given in Section 5.2.2.

Figure 5.3 shows the run time and the number of non-overlapping clusterings found in different datasets, for varying k. We observe that many non-overlapping clusters exist. This is explained by the high dimensionality of our binary data. For increasing k, the runtime and the number of clusterings found increases exponentially. A similar phenomenon occurs in traditional itemset mining: weak constraints, in our case a high value of k, lead to a combinatorial explosion where most time is spent on enumerating the solutions.

When looking at the resulting clusterings in more detail, we noticed that many of the clusterings include patterns that cover only one transaction. For $k = 3$, it is common to have one large cluster covering most of the examples, one medium sized cluster, and one cluster that covers only one transaction. Such clusterings can be considered less interesting than those in which the clusters cover about the same number of transactions. To avoid this, we introduced the optimisation settings in which we search for more balanced clusterings.

The left figures in Figure 5.4 illustrate the first optimisation setting, in which we maximise the minimum cluster size, while the right figures illustrates the second setting, in which we minimise the cluster range. In both cases, the figures show the runtime, minimum cluster size and cluster size range for different sizes k.

We see that the size of the smallest cluster decreases as the number of clusters k increases. The range of the clusters differs depending on the specific dataset. However, there is a decreasing trend in the range, indicating that a larger number of clusters can more evenly cluster the data.

To assess the influence of the types of constraints (Q3), it is also useful to compare the left and right figures in Figure 5.4. We see that the second approach, in which the range is optimised, scales less well for increasing k, although the solutions found are almost always the same. The difference can be explained by the difference between the 'minimum size' constraint and the 'size range' constraint. The minimum size constraint acts as a local frequency constraint once one candidate solution has been found. In all later solutions, each cluster has to contain at least as many examples as the smallest cluster of the earlier solution. The size range constraint, on the other hand, is a global constraint that operates on the minimum and maximum value over all clusters in the clustering, and does not reduce to a simple local frequency constraint during the search. We can conclude that if there is a choice between local and global constraints, then local constraints should be preferred as they propagate more effectively.

5.4.3 $k-$Tiling

In section 5.3.1 we presented a formulation of the area constraint and discussed why this constraint will propagate badly. Not only does it depend on $|\mathcal{I}| * |\mathcal{S}|$ variables, but each of these variables has only a weak relation to the patterns in the set. For this reason, we do not expect the $k-$tiling model to perform well, making it a good setting to investigate the limits of our CP approach (Q2).

In addition, we compare finding the optimal k-tiling to finding a greedy approximation of it. The greedy method iteratively searches for the single

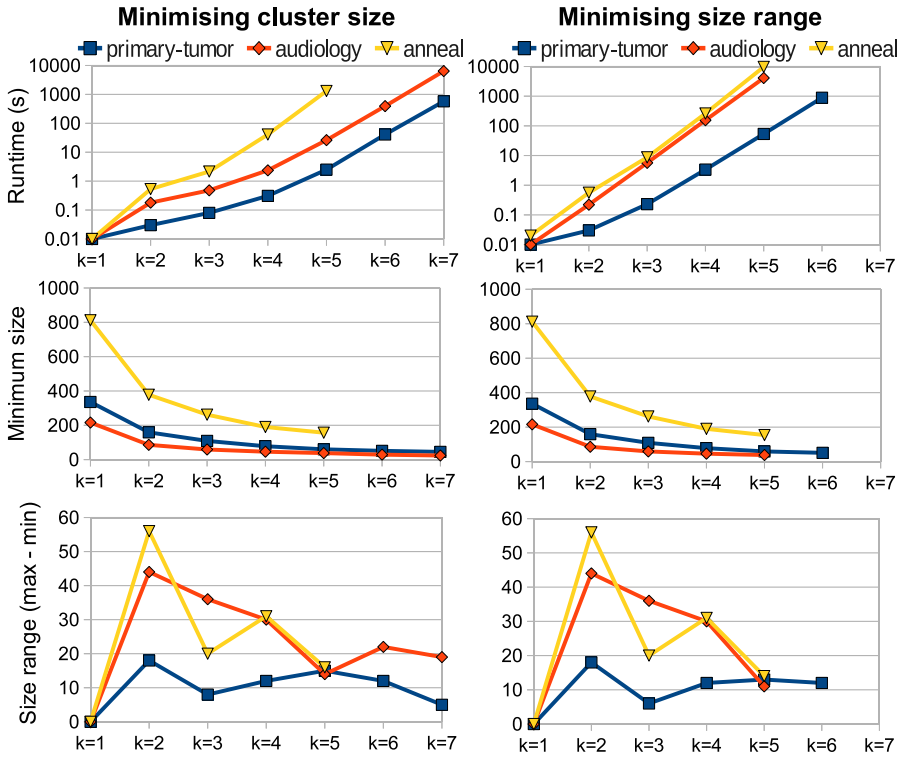


Figure 5.4: Mining a pattern set clustering with k conceptual clusters. Left: when optimising the minimum cluster size. Right: when optimising the cluster size range.

pattern (tile) that covers the largest uncovered area in the database, where we implemented the search for the best scoring tile also in the CP system; this iterative greedy algorithm is known to be an approximation of the optimum [Geerts et al. 2004]. The iterative procedure continues until k patterns are found.

Table 5.5 shows the maximum k for which an optimal k-tiling was found within a time limit of 6 hours per run, and the corresponding area. We also report the area found by a greedy k-tiling algorithm for the same value of k. Note that the highly efficient greedy algorithm always finished within 10 seconds. Only for very small values of k could an optimal k-tiling be found. Although the CP method finds the optimal solution, the greedy method's area was always close to or equal to it. In the global CP approach, the area constraint is too

	pattset k-tiling		greedy k-tiling
	max k	area for this k	area for this k
soybean	3	4447	4357
zoo	3	772	749
lymph	2	1445	1424
primary-tumor	2	1841	1841
tic-tac-toe	2	876	876
vote	2	1758	1758

Table 5.5: Maximum k and area for the k-tiling problem on multiple datasets, with a timeout of 6 hours. The right column shows the area for the same k, when using a greedy tiling algorithm.

weak to prune the search space significantly, and the search space is too big to enumerate exhaustively. This shows that for k-tiling, an exhaustive search method is not advised, at least not using the area constraint formulation of Section 5.3.1.

5.4.4 Redescription mining

We consider the case where the data consists of two partitions; they both range over the same set of transactions but consist of two different sets of items. As a reminder, we are interested in finding redescriptions, namely two patterns from the different partitions that cover many or all of the same examples. In our experimental setting, we randomly partitioned the attributes in two classes.

To investigate (Q1) we compare two approaches for finding the most frequent exact redescription. The first approach is a one step approach and models the entire problem in CP. The found patterns cover exactly the same set of examples and no other exact redescription covers more examples. In the second approach, we first find all closed itemsets that have at least one item in each partition. Redescriptions can be found by postprocessing these itemsets, as the union of two closed itemsets with equal coverage must be a closed itemset in the original data as well. Table 5.6 shows the run time needed for the one step approach (column 2) and for the first step of the two step approach (column 4). Finding all closed itemsets already takes more time than finding the global optimal solution in one step for most datasets. In the two step approach, each of the patterns would also have to be post-processed. This would increase the difference in runtime even further, especially for datasets with many solutions. Hence, on this problem the one step approach is again more promising.

	exact redescriptions		closed sets	
	time (s)	rel. freq.	time (s)	nr. solutions
hepatitis	0.08	4.38%	14.12	1824950
primary-tumor	0.02	0.60%	0.34	29395
vote	0.1	0.92%	0.4	32669
soybean	0.13	3.81%	0.11	2769
zoo	0.02	39.60%	0.03	3029

Table 5.6: Run times for the redescription mining settings; on the left, runtime and relative frequency when searching for the exact redescription covering most examples; on the right, runtime and number of patterns found when mining all closed sets forming a redescription.

To investigate (Q2) we compare the different settings for redescription mining introduced in Section 5.2.2. The first three columns of Table 5.6 list the result for the most frequent exact redescription. Figure 5.5 shows results from searching for the best redescription under different frequency thresholds, where we use the *distinct* measure as quality measure. A distinctiveness of zero corresponds to an exact rediscription. Except for the zoo dataset, the relative frequency of the redescriptions in Table 5.6 is low (column 3). For these datasets, there are no exact redescriptions covering many transactions. Figure 5.5 shows the result for different minimum frequency thresholds. Low minimum frequency thresholds lead to more similar patterns but possibly less interesting ones. Higher thresholds lead to more distinct patterns which are usually a lot more prominent in the data.

When studying the run times of the results, we can draw the following conclusions with respect to (Q3). The low run times can be attributed to the constraints at hand: the coverage and closedness local look-ahead constraints, a pairwise constraint between the two transaction sets, and a minimum frequency local look-ahead constraint for the branch-and-bound search in the first setting. The local look-ahead constraints are known to propagate well, and the pairwise constraint is able to immediately propagate changes in one transaction set to the other. Posting these constraints leads to very effective propagation and thus fast execution times. In the second setting, like the constraint for exact redescriptions, the *distinct* constraint is also a pairwise constraint. Although less effective, the fact that it is pairwise allows the constraint to effectively propagate changes between two transaction sets too.

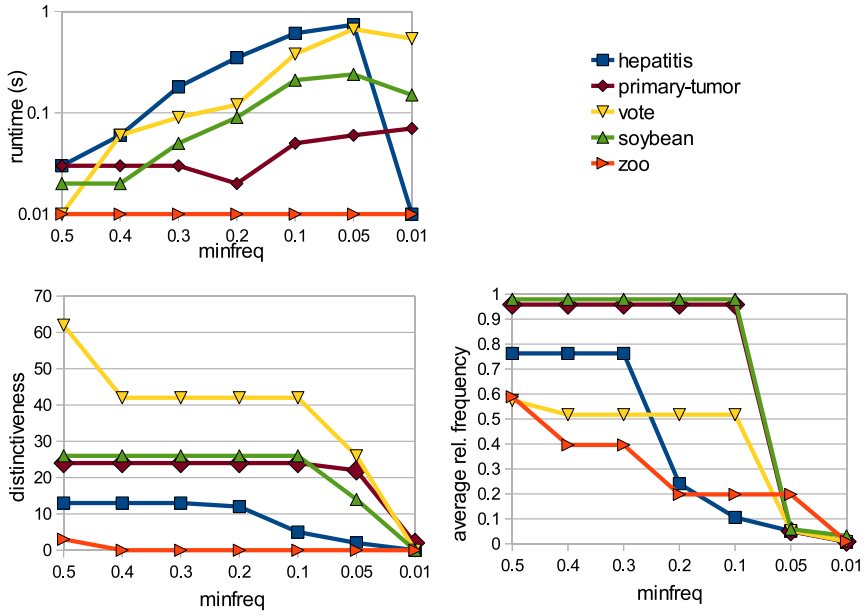


Figure 5.5: Run time, distinctiveness and average frequency of patterns when searching the least distinct redescription given a minimum frequency threshold.

5.4.5 Discussion

We have focussed our study of the suitability of CP's exhaustive search for pattern set mining on three questions.

The first question was how the proposed one step approach compared to a more traditional two step procedure. The results on the concept learning and redescription mining problems showed that a one step exhaustive search can be more efficient than a two step exhaustive search. In the two step approach the bottleneck is the large number of patterns found in the first step, in which the second step then has to search for a subset. The one step approach, on the other hand, uses the constraints of the global model to reduce the number of patterns that need to be considered.

The second question was how the k-pattern set mining approach scales to different tasks. The framework supported a great deal of tasks and different variations of them. Depending on the task at hand, there were significant differences in the runtime behaviour of the system. Even for the same task, as we saw for conceptual clustering, alternative formulations that find the same

solutions can have different runtime behaviour.

The third question was about the relation between the performance of a model and its constraints. The differences in runtime depend on the constraints used. The k-tiling problem was the task with the fewest constraints, involving the largest amount of variables. Consequently, it scaled very badly and only found solutions for the lowest values of k. Concept learning, with its better propagating *accuracy* constraint, performed better than k-tiling, although its scalability was also limited. Conceptual clustering performed better, especially when formulating it such that the optimisation constraints are local constraints. Lastly, the best results were obtained for the redescription mining task, which contained only local look-ahead and pairwise constraints.

The general question in this section was whether a generic CP approach that uses exhaustive search is also suitable for pattern set mining. We found that the answer depends on the constraints involved in the model of the mining task. As is the case in traditional pattern mining, when there are only local constraints, most of which are local-lookahead constraints, then the generic CP approach is possible. Interestingly, when there are local lookahead constraints, as well as pairwise constraints, the approach is feasible too. This was the case for the redescription mining and our models of conceptual clustering. In fact, when $k = 2$ all tasks contain only pairwise constraints. Indeed for all tasks for $k = 2$, solutions were found in an acceptable time.

In the presence of non-pairwise global constraints, the use of the proposed framework is limited to small values of k. Nonetheless, we believe that our study can lead to a better understanding of constraints, which can help the development of heuristic approaches as well. For example, the greedy approach that iteratively called the CP solver to mine for local patterns performed very well. This raises the question whether a general approach using such a *large neighbourhood search* [Shaw et al. 2002] strategy could be devised for the k-pattern set mining problem at large.

5.5 Related Work

There are two distinctive features in our approach. First, the framework for k-pattern set mining can – in contrast to most other approaches in data mining and machine learning – be used to tackle a wide variety of tasks such as classification, clustering, redescription mining and tiling. Second, our work sets itself apart by using a one step exhaustive approach, while other techniques to mining pattern sets typically use a two step approach or a heuristic one step approach.

Similar to our work is the recent work of Khiari et al. [Khiari et al. 2010] who propose to mine for n-ary patterns, patterns containing n patterns, using constraint programming. Our independently developed work goes well beyond their work by covering a much wider range of tasks and by providing a profound study on the propagation power of the constraints. In retrospect, the good efficiency results they achieved can be explained by our results: they performed experiments with either 2 or 4 patterns, in which all constraints were local-lookahead or pairwise constraints. We have observed and explained why this is essential for an exhaustive CP approach to be efficient.

We will first review local techniques to reducing redundancy in the collection of mined patterns. We then in turn discuss global exhaustive two step techniques and global heuristic techniques.

5.5.1 Local Techniques

Techniques for removing redundancy at the local level typically focus on finding patterns that form a condensed representation (Chapter 2.1.2). Condensed representations solve the redundancy problem only partly, as there are no guarantees on how much smaller the representation is compared to the full set of solutions. Condensed representations are usually mined for in the first step of typical two step algorithms. Our approach considers only closed frequent patterns. It is possible to use other condensed representations, although it could be that the globally optimal pattern set does not exist in some of the condensed representations.

5.5.2 Global Exhaustive Two step Techniques

The framework for k-pattern set mining that we introduced builds upon the notion of exhaustive pattern set mining by De Raedt and Zimmermann [De Raedt and Zimmermann 2007]. They provided a general definition of two step constraint-based pattern set mining by exploiting the analogies with local pattern mining. The key differences with the present approach is that their work assumes a two step procedure, that is, it actually is centred around the computation of

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \subseteq \mathrm{Th}(\mathcal{L}, p', \mathcal{D}) \mid p(\Pi, \mathcal{D}) \text{ is true}\}$$

in which first a local pattern mining step is performed, resulting in the set $\mathrm{Th}(\mathcal{L}, p', \mathcal{D})$ and then subsets containing such patterns are searched for. A further difference with the present approach is that we look for sets containing

a fixed number of patterns. While this is more restrictive it is – in our opinion – essential from a computational perspective. Lastly, because the approach of [De Raedt and Zimmermann 2007] is derived from local pattern mining, it suffers from the same problems as the original pattern mining algorithms, namely, an overwhelming amount of pattern sets, many of which are redundant. To avoid this problem, we focussed on finding the optimal pattern set, according to some measure, directly. By removing this optimisation criterion, our approach can be used to find all pattern sets too.

Related to the interpretation of a pattern set as a DNF formula is also the BLOSOM framework [Zhao et al. 2006], which can mine for all DNF and CNF expressions in a binary dataset. BLOSOM uses the notion of closed DNF and minimal DNF to minimise the logical redundancy in the expressions. However, a two step approach is again used in which first (variants of) frequent patterns are searched and later post-processed.

5.5.3 Global Heuristic Techniques

There are numerous heuristic approaches to finding global pattern sets that first perform a local pattern mining step and then heuristically post-process the result ([Liu et al. 1998; Bringmann and Zimmermann 2007]) (see [Bringmann et al. 2009] for an overview). Thus the second step does not guarantee that the optimal solutions are found. Furthermore, these approaches usually focus on one specific problem setting such as classification. For instance, CBA [Liu et al. 1998] first computes all frequent itemsets (with their most frequent class label) and then induces an ordered rule-list classifier by removing redundant itemsets. Several alternative techniques (for instance, [Bringmann and Zimmermann 2007; Thoma et al. 2009]) define measures of redundancy and ways to select only a limited number of patterns. Among them, greedy iterative methods are also common [Cheng et al. 2008; Thoma et al. 2009]. Constructing a concise pattern set for use in classification can be seen as a form of feature selection.

Another related problem setting is that of finding a good compression of the entire collection of frequent patterns. There exist many different approaches such as clustering the collection of frequent patterns [Yan et al. 2005], finding the patterns that best approximate the entire collection [Afrati et al. 2004], ordering the patterns such that each prefix provides a good summary of the collection [Mielikäinen and Mannila 2003], ordering according to statistical p-value [Gallo et al. 2007], and more. By construction, these techniques also work in two steps: first find all patterns given a minimum frequency threshold, then compress that collection of patterns.

Techniques also exist that try to compress the dataset rather than the collection of patterns. For example, the KRIMP algorithm [Siebes et al. 2006] uses a Minimal Description Length based global measure of compression. The compressed set of patterns covers all transactions, as we also often required in this work. Alternatively, a greedy covering technique similar to the one used in [Afrati et al. 2004] could be applied on the dataset. However, in both cases, again a heuristic algorithm is used and the size of the selected pattern set is unbounded.

In [Knobbe and Ho 2006], Knobbe and Ho present the concept of a *pattern team* as a small subset of patterns that optimises a measure. They identify a number of intuitions about pattern sets, and four measures that satisfy them; two of these are unsupervised measures while the other two are supervised. The first supervised measure uses a classifier and cross validation to assess the quality of a pattern team, which is beyond the scope of our work. The second supervised measure is area under the ROC curve. We showed in [Nijssen et al. 2009] that it is possible to exhaustively find the set of all patterns on the ROC convex hull, even without restricting the set to a size k. The work in [Knobbe and Ho 2006] differs from ours as they mostly focus on how quality measures cover certain intuitions while we focus on how to express many constraints in one framework and also on the impact of these constraints on exhaustive search.

Lastly, declarative languages for data mining tasks have been proposed before [Imielinski and Virmani 1999; Blockeel et al. 2008; Bonchi et al. 2009], usually modeled after the concept a query language for an inductive database [Imielinski and Mannila 1996]. Rather than being a query language, we propose a modeling language closer to mathematical programming, in which constraints can be decomposed, alternative formulations can be written and optimisation criteria can be expressed.

5.6 Conclusions

We introduced the k-pattern set mining problem. It tackles pattern mining directly at a global level rather than at a local one. Furthermore, it allows for the specification of many different mining tasks in a uniform way by varying the involved constraints.

We used constraint programming methodology and solvers for addressing the k-pattern set mining problem in a general way. This was realized by mapping each of the presented k-pattern set mining tasks to models in the CP framework. However, often multiple equivalent modeling strategies are possible and the model that is chosen can have a large influence on the performance of the

solver. To provide guidelines for choosing amongst alternative formulations, we categorized the individual constraints on the basis of their propagation power. Except for the natural distinction between local constraints on individual patterns and global constraints on multiple patterns, we identified two special categories: local-lookahead constraints, a type of local constraint, and pairwise constraints, which are global constraints restricted to two patterns. Constraints in these two categories propagate better than constraints in the regular local or global categories. Propagation strength is important for techniques that rely on exhaustive search such as constraint programming. Using the CP framework, we evaluated the feasibility of exhaustive search for pattern set mining. We found that a one-step search method is often more effective than a two-step method in which an exhaustive search is performed in both steps. In general, we can conclude that the feasibility of the one-step exhaustive approach depends on the constraints involved.

Nonetheless, we believe that the study of individual constraints and of how constraints interact can lead to a better understanding of the relationship between local and global constraints. In general, the study of the relation between local and global constraints, not necessarily in a one step exhaustive framework, merits further study. We hope to have contributed to this by providing a framework for rapid prototyping of existing and new problems, as well as for the controlled study of individual constraints.

There are several interesting issues for future work. First, we have presented a general problem formulation of pattern set mining and studied a number of constraints and instantiations. There is a whole range of other pattern set mining tasks and constraints that can also be studied within the k-pattern set mining framework. The usefulness of constraints, especially when using exhaustive search, depends on how well that constraint can propagate. Further study on the propagation power of constraints is required. Interesting questions include: what global constraints can be decomposed into pairwise constraints? Are better propagation algorithms possible for existing (global) constraints?

Constraint programming uses exhaustive search and the feasibility of the CP approach depended on the constraints involved. It is an open question whether other solvers can be used to solve the general k-pattern set mining problem given only its description in terms of constraints. In the artificial intelligence community, a number of constraint-based heuristic approaches have been developed that follow a similar declarative approach as constraint programming. Prominent examples include large neighbourhood search [Shaw et al. 2002] and constraint-based local search [Van Hentenryck and Michel 2009]. The authors believe such methods could provide a general solution for tasks where the CP approach currently lacks.

Chapter 6

Evaluating Pattern Set Mining Strategies using CP

This chapter[1] experimentally evaluates different search strategies for pattern set mining, using the concept learning task as representative task. The search strategies make use of the constraint programming frameworks of Chapter 3 and 5 as much as possible. The flexibility of the constraint-based frameworks allows for a systematic comparison of a diverse range of common search strategies in pattern set mining.

6.1 Introduction

Within the data mining and the machine learning literature numerous approaches exist that perform pattern set mining. These approaches employ a wide variety of search strategies. In the previous chapter, we have seen a novel one-step exhaustive strategy using constraint programming. In data mining, *step-wise* strategies are more common, where first all frequent patterns are computed and subsequently post-processed in a heuristic way; examples are CBA [Liu et al. 1998] and KRIMP [Siebes et al. 2006]. In machine learning, the *sequential covering* strategy is popular, which repeatedly and heuristically searches for a good pattern or rule and immediately adds this pattern to the

[1] Based on the paper "Evaluating pattern set mining strategies in a constraint programming framework" [Guns et al. 2011a].

current pattern- (or rule-)set; examples are FOIL [Quinlan 1990] and CN2 [Clark and Niblett 1989].

In previous chapters we studied a general constraint programming framework employing exhaustive search. We compared this approach to alternative approaches using exhaustive search. However, not all approaches in data mining use exhaustive search. The key contribution of this chapter is that we study, evaluate and compare different search strategies for pattern set mining, such as the ones mentioned above.

As it is infeasible to perform a detailed comparison on all pattern set mining tasks that have been considered in the literature, we shall focus on one prototypical task from the previous chapter: concept learning. In this task, the aim is find the most accurate description of a concept for which positive and negative examples are given. A description is here a set of boolean formulas, also called patterns. Our focus is on the exploration of a wide variety of search strategies, from greedy to complete and from step-wise to one-step approaches.

To be able to obtain a fair and detailed comparison we choose to reformulate the different strategies within the common framework of constraint programming. In the previous chapters we have shown that constraint programming is a very flexible and usable approach for tackling a wide variety of local pattern mining tasks (such as closed frequent itemset mining and discriminative itemset mining) as well as for solving different k-pattern set mining tasks. In this chapter, we employ the constraint programming framework to compare different search strategies for pattern set mining, focusing on the boolean concept learning task.

This chapter is organized as follows: in Section 6.2, we briefly remind the reader of the pattern set mining problem and the concept learning task; in Section 6.3, we introduce various search strategies for pattern set mining, each using the constraint programming framework in a different way; in Section 6.4, we report on experiments, and finally, in Section 6.5, we conclude.

6.2 Pattern Set Mining Task

The benchmark task on which we shall evaluate different pattern set mining strategies is that of finding boolean concepts in the form of k-term DNF expressions, also known as concept learning. This task is well-known in computational learning theory [Kearns and Vazirani 1994] and is closely related to rule-learning systems such as FOIL [Quinlan 1990] and CN2 [Clark and

Niblett 1989] and data mining systems such as CBA [Liu et al. 1998] and KRIMP [Siebes et al. 2006].

We previously introduced this as the following pattern set mining problem:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{accuracy}\left(\mathit{freq}^{+}(\Pi), \mathit{freq}^{-}(\Pi)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{closed}^{+}(\pi), \end{aligned}$$

where Π is a set of k patterns; $\Pi = \{\pi_1, \ldots, \pi_k\}$.

Finding a good pattern set is often a hard task; many pattern set mining tasks, such as the task of k-term DNF learning, are NP complete [Kearns and Vazirani 1994]. Hence, there are no straightforward algorithms for solving such tasks in general, giving rise to a wide variety of search algorithms. The strategies they employ can be categorized along two dimensions.

Two-Step vs One-Step: in the two-step approach, one first mines patterns under local constraints to compute the set $\mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$; afterwards, these patterns are fed into another algorithm that computes $\arg\max_{\Pi \in \mathrm{Th}(\mathcal{L}, p, \mathcal{D})} f(\Pi)$ using post-processing. In the one-step approach, this strict distinction between the two phases cannot be made.

Exact vs Approximate: exact methods provide strong guarantees for finding the optimal pattern set under the given constraints, while approximate methods employ heuristics to find good though not necessarily optimal solutions.

In the next section we consider different instantiations of these settings for the task of concept learning.

6.3 Constraint Programming Model

Using the constraints of Section 5.2.1 we now present the declarative specifications of two-step and one-step strategies, for both the exact and approximate cases.

6.3.1 Two-Step Pattern Set Mining

In the first step, a collection of local patterns is mined, while in the second step, a small number of patterns is selected from that collection. We describe different strategies for each step in turn.

Step 1: Local Pattern Mining.

Using the constraint programming framework one can formulate many local pattern mining problems, such as frequent and discriminative pattern mining. Indeed, consider the following constraints:

$$\forall t \in \mathcal{S}: \quad T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0. \qquad (\textit{Coverage})$$

$$\forall i \in \mathcal{I}: \quad I_i = 1 \leftrightarrow \sum_{t \in \mathcal{S}} T_t(1 - \mathcal{D}_{ti}) = 0. \qquad (\textit{Closedness})$$

$$\forall i \in \mathcal{I}: \quad I_i = 1 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} \geq \theta. \qquad (\textit{Min. frequency})$$

$$\forall i \in \mathcal{I}: \quad I_i = 1 \rightarrow \text{accuracy}(\sum_{t \in \mathcal{S}^+} T_t \mathcal{D}_{ti}, \sum_{t \in \mathcal{S}^-} T_t \mathcal{D}_{ti}) \geq \theta. \qquad (\textit{Min. accuracy})$$

In these constraints, the *coverage constraint* links the items to the transactions; the *closedness constraint* removes redundancy by ensuring that an itemset has no superset with the same frequency. A closed-on-the-positives constraint as in Section 5.2.2 would be sufficient here. However, we prefer to emulate a traditional two-step approach. These employ a standard closed itemset mining algorithm in the first step. Mining only for closed-on-the-positive itemsets is most likely to reduce the runtime of the approach, but the outcome would remain the same. The *minimum frequency constraint* ensures that itemset I covers at least θ transactions. Alternatively, to mine for all accurate patterns instead of all frequent patterns, the *minimum accuracy constraint* can be used, which ensures itemsets have an accuracy of at least θ.

To emulate the first step of two-step approaches that are common in data mining [Liu et al. 1998; Siebes et al. 2006; Bringmann et al. 2010], we shall employ two alternatives: 1) using **frequent closed patterns**, which are found with the *coverage*, *closedness* and *minimum frequency* constraints; 2) using **accurate closed patterns**, found with the *coverage*, *closedness* and *minimum accuracy* constraints. Both of these approaches preform the first step in an *exact* manner. They find the set of all local patterns adhering to the constraints.

Step 2: Post-processing the Local Patterns.

Once the local patterns have been computed, the two-step approach post-processes them in order to arrive at the pattern set. We describe the two main strategies for this.

Post-processing by Sequential Covering (Approximate). The most simple approach to the second step is to perform greedy sequential covering, in which one iteratively selects the best local pattern from $\mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$, according to the optimisation measure f. In case of the accuracy measure, all examples that are covered are then removed from the data. This continues until the desired number of patterns k has been reached or until all positive examples have been covered. This type of approach is most common in data mining systems. Using greedy sequential covering results in an approximation of $\arg\max_{\Pi \in \mathrm{Th}(\mathcal{L},p,\mathcal{D})} f(\Pi)$, even though the local patterns were mined exhaustively.

Post-processing using Complete Search (Exact). Another possibility is to perform a new round of pattern mining, this time on the collection of local patterns as described in [De Raedt and Zimmermann 2007]. In this case, each previously found pattern in $\mathcal{P} = \mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$ can be seen as an item r in a new database; each new item identifies a pattern. One is looking for the set of pattern identifiers $P \subseteq \mathcal{P}$ with the highest accuracy. In this case, the set is not a conjunction of items, but a disjunction of patterns, meaning that a transaction is covered if at least one of the patterns $r \in P$ covers it. We studied how this can be formalized in a constraint programming framework in Section 5.3.2, using a *disjunctive coverage constraint*, a *minimum accuracy constraint* and a *size constraint*:

$$\forall t \in \mathcal{S} : T_t = 1 \leftrightarrow \sum_{r \in \mathcal{P}} P_r \mathcal{M}_{ti} > 0, \qquad (\textit{Disj. coverage})$$

$$\textit{accuracy}\Big(\sum_{t \in \mathcal{S}^+} T_t, \sum_{t \in \mathcal{S}^-} T_t\Big) \geq \theta, \qquad (\textit{Accurate})$$

$$\sum_{r \in \mathcal{P}} P_r = k, \qquad (\textit{Size of set})$$

where $\mathcal{M}$ is a data matrix in which the columns correspond to the patterns in $\mathcal{P}$, the rows correspond to the transactions in $\mathcal{S}$ and the elements in the matrix indicate which pattern covers which transaction. A solution is a set of k patterns from $\mathcal{P} = \mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$ that together cover the transactions most accurately.

This type of exact two-step approach is relatively new in data mining. Two notable works are [Knobbe and Ho 2006; De Raedt and Zimmermann 2007]. In these publications, it was proposed to post-process a set of patterns by using a complete search over subsets of patterns. If an exact pattern mining algorithm is used to compute the initial set of pattern in the first step, this gives a method that is overall exact and offers strong guarantees on the quality of the solution found.

6.3.2 One-Step Pattern Set Mining

This type of strategy, which is common in machine learning, searches for the pattern set $\mathrm{Th}(\mathcal{L}, p, \mathcal{D})$ directly, that is, the computation of $\mathrm{Th}(\mathcal{L}, p_{local}, \mathcal{D})$ and $\mathrm{Th}(\mathcal{L}, p, \mathcal{D})$ using p_{global} is integrated or interleaved. This can remove the need to have strong constraints with strict thresholds in p_{local}. There are two approaches to this:

Iterative Greedy Sequential Covering (Approximate). In the iterative greedy sequential covering approach that we investigate here, an iterative greedy search is employed to heuristically find the best pattern set. At each iteration, a local pattern mining algorithm is used to find the best local pattern (with the highest accuracy). This pattern is greedily added to the pattern set. All examples covered are then removed from the data, and a new iteration begins. This setting is similar to 2-step sequential covering, only that here, at each iteration, the most accurate pattern is mined for directly, instead of selecting it from a set of previously mined patterns. Mining for the most accurate pattern can be done in a constraint programming setting by doing branch-and-bound search.

Examples of one-step greedy sequential covering methods are FOIL and CN2; however, they also use a greedy algorithm to find the local pattern in each iteration. We, instead, use an exact branch-and-bound pattern miner in each iteration of the greedy sequential covering.

Iterative Sequential Covering with Beam Search (Approximate). This setting is the same as the one above, except that in each iteration one searches for the top-b patterns with the highest accuracy, where b is the beam width. For each of the top-b patterns, sequential covering is again performed with a beam width of b. From all those solutions, only the b best pattern sets found are used in the next iteration. After k iterations, the best pattern set is returned.

The use of branch-and-bound pattern mining for identifying top-b patterns has been previously studied in the data mining literature; see for instance

[Bringmann and Zimmermann 2005]. In the experimental section, we shall consider different versions of the approach, corresponding to different sizes of the beam.

Global Optimization (Exact). The last option is to specify the problem of finding a k-pattern set as a single global optimization problem. This was explained in the previous chapter, see Section 5.3.2 for the one-step formulation of the concept learning task.

The one-step global optimization approaches to pattern set mining are less common; we have detailed such an approach in Chapter 5 and are further only aware of the work of Khiari et al. [Khiari et al. 2010]. One could argue that some iterative pattern mining strategies will find pattern sets that are optimal under certain conditions. For instance, "Tree^2" [Bringmann and Zimmermann 2005] can find a pattern set with minimal error on supervised training data; however, it neither provides guarantees on the size of the final pattern set nor provides guarantees under additional constraints.

6.4 Experiments

On the task of concept learning, we now compare the different search strategies presented and answer the following two questions:

- Q1: Under what conditions do the different strategies perform well?
- Q2: What quality/runtime trade-offs do the strategies make?

We measure how well a pattern set describes the target concept by evaluating its accuracy on a dataset. In a predicitive setting, one would have to do the evaluation on a separate test set as the goal is to learn a hypothesis that generalizes to an underlying distribution. However, in the boolean concept learning task we consider, the accuracy on the whole dataset is an appropriate measure as the goal is to find a good description of that particular data.

The experiments were performed using the Gecode-based framework of Chapter 3 and performed on PCs running Ubuntu 8.04 with Intel(R) Core(TM)2 Quad CPU Q9550 processors and 4GB of RAM. The datasets[2] were derived from the UCI Machine Learning repository [Frank and Asuncion 2010] by discretising numeric attributes into eight equal-frequency bins. To obtain

[2] http://dtai.cs.kuleuven.be/CP4IM/datasets/

	Mushroom	Vote	Hepatitis	Germ-cr.	Austr-cr.	Kr-vs-kp
Transactions	8124	435	137	1000	653	3196
Items	119	48	68	112	125	73
Class distr.	52%	61%	81%	70%	55%	52%
Total patterns	221524	227032	3788342	25M+	25M+	25M+
Patt. poor/rich	poor	poor	poor	rich	rich	rich
frequency ≥ 0.7	12	1	137	132	274	23992
frequency ≥ 0.5	44	13	3351	2031	8237	369415
frequency ≥ 0.3	293	627	93397	34883	257960	25M+
frequency ≥ 0.1	3287	35771	1827264	2080153	24208803	25M+
accuracy ≥ 0.7	197	193	361	2	11009	52573
accuracy ≥ 0.6	757	1509	3459	262	492337	2261427
accuracy ≥ 0.5	11673	9848	31581	6894	25M+	25M+
accuracy ≥ 0.4	221036	105579	221714	228975	25M+	25M+

Table 6.1: Data properties and number of patterns found for different constraints and thresholds. 25M+ denotes that more than 25 million patterns were found.

reasonably balanced class sizes we used the majority class as the positive class. Experiments were run on many datasets, but we here present the findings on 6 diverse datasets whose basic properties are listed in the top 3 rows of Table 6.1.

6.4.1 Two-Step Pattern Set Mining

The result of a two-step approach obviously depends on the quality of the patterns found in the first step. We start by investigating the feasibility of this first step, and then study the two-step methods as a whole.

Step 1: Local Pattern Mining.

As indicated in Section 6.3.1, we employ two alternatives: using frequent closed patterns and using accurate closed patterns. Both methods rely on a threshold to influence the number of patterns found.

Table 6.1 lists the number of patterns found on a number of datasets, for the two alternatives and with different thresholds. Out of practical considerations we stopped the mining process when more than 25 million patterns were found. Using this cut-off, we can distinguish pattern poor data (data having less than 25 million patterns when mining unconstrained) and pattern rich data. In the case of pattern poor data, one can mine using very low or even no thresholds.

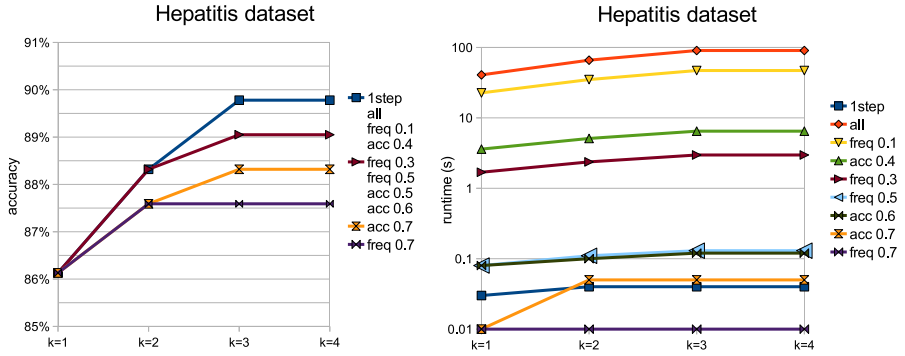


Figure 6.1: Quality & runtime for approximation methods, pattern poor hepatitis dataset. In the left figure, *1step, all, freq 0.1, acc 0.4* and *freq 0.3, freq 0.5, acc 0.5, acc 0.8* are each grouped together because they attain the same score.

In the case of pattern rich data, however, one has to use a more stringent threshold in order not be overwhelmed by patterns. Unfortunately, one has to mine with different thresholds to discover how pattern poor or rich an unseen dataset is.

Step 2: Post-processing the Local Patterns.

We now investigate how the quality of the global pattern sets is influenced by the threshold used in the first step, and how this compares to pattern sets found by 1-step methods that do not have such thresholds.

Post-processing by Sequential Covering (Approximate). This two-step approach iteratively picks the best local pattern from the set of patterns computed in step one. As such, the quality of the pattern set depends on whether the right patterns are in the pre-computed pattern set. We use our generic framework to compare two-step sequential covering to the one-step approach.

For pattern poor data for which the set of all patterns can be calculated, such as the mushroom, vote and hepatitis dataset, using all patterns obviously results in the same pattern set as found by the one-step approach. Figure 6.1 shows the prototypical result for such data: low thresholds lead to good pattern sets, while higher thresholds gradually worsen the solution. For this dataset, starting from

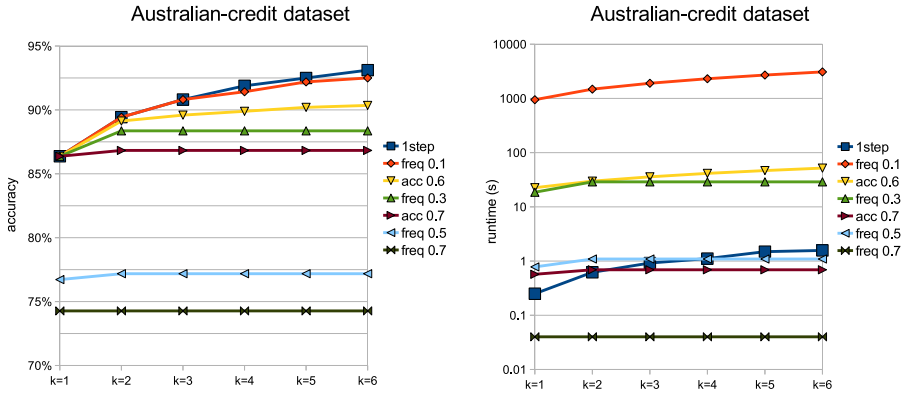


Figure 6.2: Quality & runtime for approximation methods, pattern rich australian-credit dataset.

$k = 3$, no better pattern set can be found as the first 3 patterns already cover all positive examples. The same is true for the mushroom dataset, while in the vote dataset the sequential covering method continues to improve for higher k. Also note that in Figure 6.1 a better solution is found when using patterns with accuracy greater than 40%, compared to patterns with accuracy greater than 50%. This implies that a better pattern set can be found containing a local pattern that has a low accuracy on the whole data. This indicates that using accurate local patterns does not permit putting high thresholds in the first step.

With respect to question Q2, we can observe that using a lower threshold comes at the cost of higher runtimes. However, for pattern poor datasets such as the one in Figure 6.1, these times are still manageable. The remarkable efficiency of the one-step sequential covering method is thanks to the effective handling of discriminative constraints, discussed in Section 3.4.

On pattern rich data such as the german-credit, australian-credit and kr-vs-kp dataset, similar behaviour can be observed. The only difference is that one is forced to use more stringent thresholds. Because of this, the pattern set found by the one-step approach can usually not be found by the two-step approaches. Figure 6.2 exemplifies this for the australian-credit dataset. Using a frequency threshold of 0.1, the same pattern set as for the one-step method is found for up to $k = 3$, but not so for higher k. When using the highest frequency thresholds, there is a risk of finding significantly worse pattern sets. On the kr-vs-kp dataset, when using high frequency thresholds significantly worse results were found as well, while this was not the case for the accuracy

	Mushroom		Vote		Hepatitis		German-cr.		Austr-cr.	
	k	sec	k	sec	k	sec	k	sec	k	sec
all	–		–		–		-		-	
freq. ≥ 0.7	6	0.2	only 1 pat		6	0.03	6	2.12	6	0.59
freq. ≥ 0.5	6	2.2	6	0.01	2	2650	2	8163	6	14244
freq. ≥ 0.3	6	14	6	0.89	–		–		–	
freq. ≥ 0.1	2	9477	1	1015	–		–		–	
acc. ≥ 0.7	6	8.6	6	0.12	6	3.05	6	0.01	1	713
acc. ≥ 0.6	*4	6714	5	14205	2	6696	6	104	–	
acc. ≥ 0.5	–		1	391	1	3169	1	696	-	
acc. ≥ 0.4	–		–		–		–		-	

Table 6.2: Largest k (up to 6) and time to find it for the 2-step complete search method. – indicates that step 2 did not manage to finish within the timeout of 6 hours, for the kr-vs-kp datasets (not shown) this was the case in all settings. - indicates that step 1 was aborted because more than 25 million patterns were found. * indicates that no other method found a better pattern set.

threshold. With respect to Q2 we have again observed that lower thresholds lead to higher runtimes for the two-step approaches. Lowering the thresholds further to find even better pattern sets would correspondingly come at the cost of even higher computation times.

Post-processing using Complete Search (Exact). When post-processing a collection of patterns using complete search, the size of that collection becomes a determining factor for the success of the method. Table 6.2 shows the same datasets and threshold values as in Table 6.1; here the entries show the largest k for which a pattern set could be found, up to $k = 6$, and the time it took. A general trend is that in case many patterns are found in step 1, e.g., more than 100 000, the method is not able to find the optimal solution.

With respect to Q1, in further experimentation we discovered that only for the mushroom dataset the method found a better pattern set than any other method. This result was obtained with a frequency threshold of 0.4, compared to all other method/threshold combinations. For all other sets found when post-processing using complete search, one of the 1-step methods found a better solution. Hence, although this method is exact in its second step, it depends on good patterns from its first step. Unfortunately finding those usually requires using low threshold values with corresponding disadvantages.

Mushroom	Vote	Hepatitis	German-credit	Australian-credit	Kr-vs-kp
$k = 2$	$k = 4$	$k = 3$	$k = 2$	$k = 2$	$k = 3$

Table 6.3: Largest k for which the optimal solution was found within 6 hours.

6.4.2 One-Step Pattern Set Mining

In this section we compare the different one-step approaches that do not impose a threshold on local patterns. We investigate how feasible the one-step exact approach is, as well as how close the greedy sequential covering method brings us to this optimal solution, and whether beam search can close the gap between the two.

When comparing the two-step sequential covering approach with the one-step approach, we already observed that the latter is very efficient, though it might not find the optimal solution. The one-step exact method is guaranteed to find the optimal solution, but has a much higher computational cost. Table 6.3 shows up to which k the exact method was able to find the optimal solution within the 6 hours time out. Comparing these results to the two-step exact approach in Table 6.2, we see that pattern sets can be found directly where the two-step approach failed, even when using certain thresholds.

With respect to Q1 we observed that only for the kr-vs-kp dataset the greedy method, and hence all beam searches with a larger beam, found the same pattern sets as the exact method. For the mushroom and vote dataset, starting from beam width 5, the optimal pattern set was found. For the german-credit and australian-credit, a beam width of size 15 was necessary. The hepatitis dataset was the only dataset for which the complete method was able to find a better pattern set, in this case for $k = 3$, within the timeout of 6 hours.

Figure 6.3 shows a representative figure, in this case for the german-credit dataset: while the greedy method is not capable of finding the optimal pattern set, larger beams successfully find the optimum. For $k = 6$, beam sizes of 15 or 20 lead to a better pattern set than when using a lower beam size. The exact method stands out as being the most time consuming. For beam search methods, larger beams clearly lead to larger runtimes. The runtime only increases slightly for increasing sizes of k because the beam search is used in a sequential covering loop that shrinks the dataset at each iteration.

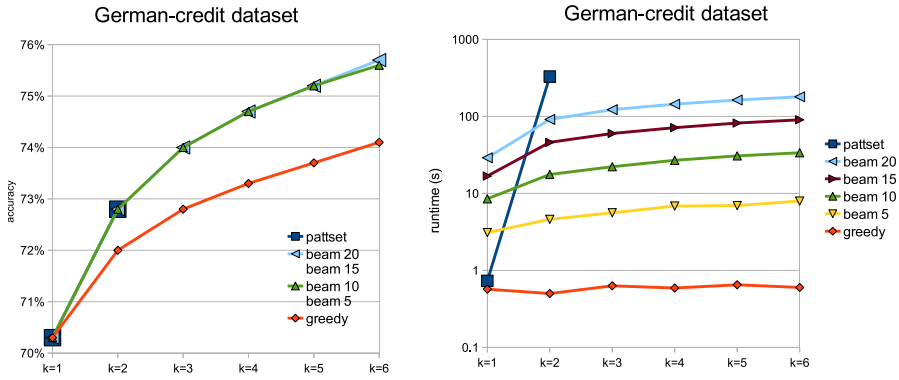


Figure 6.3: Quality & runtime for 1-step methods, german-credit dataset. In the left figure, *beam 20, beam 15* and *beam 10, beam 5* are each grouped together because they attain the same score. *pattset* and the *beam 20, beam 15* group only differ in score for $k = 8$.

6.5 Conclusions

We compared several methods for finding pattern sets within a common constraint programming framework, where we focused on boolean concept learning as a benchmark. We distinguished one-step from two-step approaches, as well as exact from approximate ones. Each method has its strong and weak points, but the one-step approximate approaches, which iteratively mine for patterns, provided the best trade-off between runtime and accuracy. The method does not depend on a threshold, unlike two-step methods that need to add constraints out of practical considerations, to limit the result of the first step. Additionally, one-step approximate approaches can easily be improved by using a beam search strategy. The exact approaches, perhaps unsurprisingly, do not scale well to larger and pattern-rich datasets. A newly introduced approach for one-step exact pattern set mining however has optimality guarantees and performs better than previously used two-step exact approaches. In future work our study can be extended to consider other problem settings in pattern set mining, as well as other heuristics and evaluation metrics; furthermore, even though we cast all settings in one implementation framework in this chapter, a more elaborate study could clarify how this approach compares to the pattern set mining systems in the literature.

Chapter 7

cis-Regulatory Module Detection using CP

This chapter[1] applies the constraint programming for itemset mining framework of Chapter 3 on a challenging bioinformatics problem. This requires the addition of domain-specific knowledge and constraints. The chapter demonstrates the flexibility of the constraint programming approach, even towards non-standard, complex constraints.

7.1 Introduction

Gene expression is the process through which cells respond to changes in their environment. Expression of genes is influenced by regulatory elements such as transcription factors. *Transcription factors* (TFs) can bind on the DNA sequence upstream of a gene, thereby regulating its expression. This is done by activating or inhibiting the transcription of DNA to mRNA. Transcription factors are known to bind only on specific subsequences of the DNA, called transcription factor binding sites. Often, gene expression is not regulated by a single TF, but rather by a set of TFs binding in each other's proximity on the sequence. Such a cluster of transcription factor binding sites is called a *cis-regulatory module* (CRM). The detection of CRMs is key in developing a

[1] Based on the paper "Cis-regulatory module detection using constraint programming" [Guns et al. 2010b].

better understanding of the regulation of the genome and the interaction of cells and viruses with their environment.

Many tools have been developed to aid the bio-informatician in discovering regulatory elements in DNA sequences. An important class of such tools deals with the problem of finding transcription factor binding sites in a single given sequence, see for example [Tompa et al. 2005] for an overview and comparison. Although successful to a certain degree, the performance of these methods suffers from the prediction of many false positive binding sites [Tompa et al. 2005]. Key in avoiding these false positives is the exploitation of additional knowledge, if available.

In this chapter we study one such case, in which we assume the upstream sequence of several co-regulated genes is given. Additionally, we start from known sequence motifs of transcription factors. A *motif* represents a model of the known binding sites of a transcription factor. Known transcription factor binding sites, and hence motif models of them, are made available through databases such as TRANSFAC [Matys et al. 2003] and JASPAR [Sandelin et al. 2004]. Starting from a large collection of motifs, we are interested in finding a set of motifs having *cis*-regulatory modules (CRMs) in multiple sequences. There are several features that could allow us to reduce the number of false positive predictions in this case:

- given a set of co-regulated sequences, the goal is to find the set of motifs that regulate them. As biological data is inherently noisy, we can not require that the set of motifs has a CRM in each co-regulated sequence. However, motifs with CRMs in only a small fraction of the sequences can be excluded;
- the binding sites constituting a CRM are typically clustered together on the sequence. When detecting CRMs, it is sufficient to consider only sites that bind in each other's proximity. This is common practice in CRM detection tools [Sharan et al. 2003; Frith et al. 2003; Aerts et al. 2003; Sandve et al. 2008];
- the identified CRMs can be statistically assessed using independent genomic sequences. A large number of such sequences are publicly available; we will select such sequences to serve as a *background model* in our statistical evaluation.

Ideally, we would like to exploit all of the above properties in one algorithm. However, few algorithms are capable of this. Several techniques have been proposed for the discovery of *cis*-regulatory modules [Sharan et al. 2003; Frith et al. 2003; Aerts et al. 2004], but they do not consider multiple sequences.

Other tools do not take into account the proximity of binding sites [Sun et al. 2009; Turi et al. 2009], or differ in the evaluation of the CRMs [Sandve et al. 2008].

We propose a general approach which takes all three properties described above into account. The approach is based on our constraint programming for itemset mining framework. *cis*-Regulatory module detection can be seen as a constraint-based itemset mining task: a set of motifs corresponds to an itemset, where each motif corresponds to a single item. The genomic sequences correspond to transactions, where each motif possibly binds to the 'transaction' or not. The constraints will enforce that a solution satisfies the three domain-specific properties described above.

Itemset mining has been used for CRM detection before [Sun et al. 2009; Turi et al. 2009; Sandve et al. 2008], but our technique differs in several ways from these earlier techniques. First, most itemset mining based techniques do not take into account proximity of binding sites [Sun et al. 2009; Turi et al. 2009]. Second, we use a novel approach to eliminate redundant sets of motifs; this makes our algorithm more efficient and renders its output more useful. It also allows us to deal with both large numbers of motifs and large numbers of sequences. Finally, in order to be able to deal with this wide range of requirements, we build on the general principles of constraint programming, which allows us to combine these requirements more flexibly.

We will first introduce the general idea of our approach, followed by a discussion of how the constraint programming for itemset mining framework can be extended to CRM detection. Finally, our system called CPModule is validated by experiments.

7.2 Method Overview

Our method consists of 3 phases: screening, mining and ranking. In the remainder of this section we provide an overview of the tasks performed in these phases. Our main contribution is in the calculations performed in the mining phase. Details of this phase are provided in the section following this one.

7.2.1 Phase 1: Screening

Like most other module detection methods [Frith et al. 2001, 2003; Aerts et al. 2004; Sandve et al. 2008], our method starts from an existing library of motif

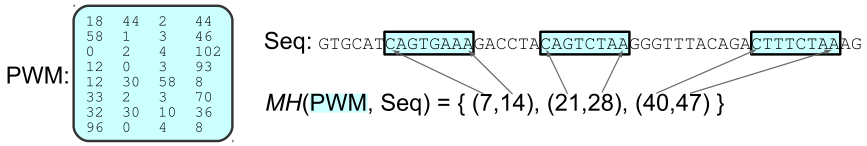


Figure 7.1: Example PWM and example motif hits on a given sequence.

models for transcription factors, in this case position weight matrices (PWMs). The *motif* of a transcription factor is a representation of the DNA segments where the transcription factor is known to bind. A position weight matrix is a fixed-length matrix representation that has four columns, one for each base A, C, G and T, and as many rows as positions of a sequence it can match. A PWM can be used to score a DNA segment of the right length, indicating how well the PWM matches that segment. *Screening* is the process of matching a PWM with all possible segments of a sequence. A segment with a high score is called a *hit* and is a putative binding site for the transcription factor. By applying a threshold on the score, we can identify none, one or multiple hits per motif and per genomic sequence. The result of the screening phase is for each motif M and sequence S a set of motif hits

$$MH(M,S) = \{(l,r) \,\big|\, 1 \leq l < r \leq |S|, M \text{ has a hit at } (l,r)\};$$

here (l,r) is an interval between positions l and r on the sequence. Figure 7.1 shows an example PWM and sequence, as well as three example hits. To identify hits, many alternative systems can be used; we will use the Clover tool [Frith et al. 2004].

7.2.2 Phase 2: Mining

This phase takes as input the motif hits in the target sequences, found during the screening phase. We use these hits to search for potential CRMs across multiple sequences. A potential CRM is characterized by a set of motif hits appearing in each other's proximity on a sequence. If there is a region in a sequence in which each of the motifs in the motif set has at least one hit, we say that they are in each other's proximity on that sequence. The maximal distance of a region is specified by the user and controls the level of proximity. More formally, a set of motifs $\mathcal{M} = \{M_1, \ldots M_n\}$ is a potential CRM in a sequence S iff its set of hit regions with maximum distance δ is not empty,

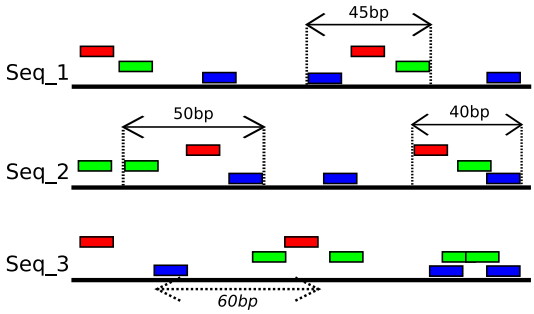


Figure 7.2: Proximity of a motif set in 3 genomic sequences (distance = 50 base pairs). Each thick black line represents a genomic sequence. Each color represents a different motif and each box a hit of that motif on that sequence.

where this set of hit regions is defined as follows:

$$HR(\mathcal{M}, S) = \{(l, l+\delta) \,\big|\, 1 \leq l \leq |S|, \forall M \in \mathcal{M} : \exists (l', r') \in MH(M, S) : l \leq l' < r' \leq l + \delta\}.$$

An example is shown in Figure 7.2. Given a maximum distance of 50bp, the Red, Green and Blue motifs are within each other's proximity in sequence 1 and 2. In sequence 2, there are two regions containing hits of the 3 motifs. In sequence 3, the smallest region containing all 3 motifs has a distance of 60bp. Hence according to the threshold, the motifs are not within each other's proximity in that sequence.

Given a set of sequences $\mathcal{S}$, the subset of sequences in which a set of motifs $\mathcal{M}$ forms a potential CRM is denoted by $\varphi(\mathcal{M}, \mathcal{S})$.

$$\varphi(\mathcal{M}, \mathcal{S}) = \{S \in \mathcal{S} \,|\, HR(\mathcal{M}, S) \neq \emptyset\}$$

$\varphi(\mathcal{M}, \mathcal{S})$ is a coverage function for the CRM detection problem, like $\varphi_{\mathcal{D}}(I)$ is for itemset mining.

A key element of our approach is that we are looking for representative motif sets across multiple sequences. We formalize this by requiring that a minimal number of the input sequences contains the motif set as a potential CRM, i.e. $|\varphi(\mathcal{M}, \mathcal{S})| \geq \theta$, for some minimum frequency threshold θ. Hence our mining algorithm enumerates all combinations of motifs that have hits in each other's proximity, in a sufficient number of genomic sequences. We will use our constraint programming for itemset mining framework to solve this task.

The output of this phase is hence a list of motif sets $\mathcal{M}$, each a potential CRM. An important issue is that of redundancy in the collection of motif sets. It is possible that two motif sets $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1 \subset \mathcal{M}_2$, are found in the same set of sequences, i.e. $\varphi(\mathcal{M}_1, \mathcal{S}) = \varphi(\mathcal{M}_2, \mathcal{S})$. To improve the efficiency of the search and reduce the size of the input of the next phase, it can be desirable not to list all redundant sets. Our framework allows us to deal with this issue by enforcing a *closedness* constraint (see Section 3.3).

7.2.3 Phase 3: Ranking

To assess the significance of each of the potential CRMs found in the previous phase, we determine their statistical significance. We want to find motif sets which are very specific to our target genomic sequences, but not to a background model [Van Loo et al. 2008]. The background model consists of a large number of random samples of sequences from the genome. We use it to calculate a p-value, and rank the potential CRMs accordingly.

To calculate the p-value, we adapt the strategy proposed in [Gallo et al. 2007]. We compare the number of observed sequences that contain the motif set, $|\varphi(\mathcal{M}, \mathcal{S})|$, with the expected number of sequences. The latter is estimated by counting the number of background sequences containing the motif set, where we use exactly the same screening as on the target sequences to calculate $|\varphi(\mathcal{M}, \mathcal{S}_{background})|$. From this, we calculate a p-value by means of a binomial distribution:

$$p - value(\mathcal{M}) = \sum_{i=|\varphi(\mathcal{M},\mathcal{S})|}^{|\mathcal{S}|} \binom{|\mathcal{S}|}{i} p^i (1-p)^{|\mathcal{S}|-i};$$

where $p = |\varphi(\mathcal{M}, \mathcal{S}_{background})|/|\mathcal{S}_{background}|$; $\mathcal{S}$ is the set of target sequences; $\mathcal{S}_{background}$ the set of background sequences.

Note that in this ranking, for two motif sets $\mathcal{M}_1 \subseteq \mathcal{M}_2$, with $\varphi(\mathcal{M}_1, \mathcal{S}) = \varphi(\mathcal{M}_2, \mathcal{S})$, motif set $\mathcal{M}_2$ will never score worse than $\mathcal{M}_1$ and is hence of more interest. Avoiding redundancies in phase 2 will prevent patterns like $\mathcal{M}_1$ to clutter the ranking.

7.3 Constraint Programming Model

As pointed out in the previous section, we need to find a set of motifs fulfilling a wide range of conditions. To solve this combinatorial problem, in previous

work specialised itemset mining algorithms have been used [Sun et al. 2009; Turi et al. 2009; Sandve et al. 2008]. To address the method of the previous section, we could also develop a specialized itemset mining algorithm. The main drawback of such an approach would be that it does not provide indications on how to extend the algorithm in the future if further extensions are needed. Hence we propose to use a more general approach, based on the constraint programming for itemset mining framework used in the previous chapters. Its modular approach also allows us to combine the constraints we are currently faced with in a principled and correct way.

In the following, we propose a constraint programming model for the CRM detection problem, consisting of 4 different constraints which we describe in turn. We exploit the modularity of CP by adding a new constraint to our framework, which is sufficient to deal with the proximity requirement of our CRM detection method.

We propose a CSP formulation in which there is a boolean variable $\widetilde{M}_i$ for every motif (corresponding to an item in the itemset mining formulation). The variable $\widetilde{M}_i$ indicates whether this motif is part of the motif set. If a certain $\widetilde{M}_i = 1$, then we say that the motif (c.f. item) is in the motif set (cf. itemset); otherwise the motif is not in the set. Hence, $\mathcal{M} = \{M_i \mid \widetilde{M}_i = 1\}$. Furthermore, we have a boolean variable $\widetilde{S}_j$ for every genomic sequence (corresponding to a transaction), indicating whether the motif set is a potential CRM in a sequence, i.e. whether $S_j \in \varphi(\mathcal{M}, \mathcal{S})$. Lastly, we define a boolean variable $\widetilde{seqM}_{ij}$ for every motif i and every sequence j. The variables $\widetilde{seqM}_{ij}$ indicate whether motif M_i is in the proximity of the motifs in motif set $\mathcal{M}$ on sequence j. This corresponds to the boolean matrix $\mathcal{D}$ in the itemset mining formulation, as explained in more detail below.

The following constraints are imposed on these variables:

Proximity Constraint The essential constraint for this task is the proximity constraint, which will determine the truth value of the $\widetilde{seqM}_{ij}$ variables. Formally, we define the $\widetilde{seqM}_{ij}$ variables as follows for every motif on every genomic sequence:

$$\forall ij : \widetilde{seqM}_{ij} = 1 \Leftrightarrow (\exists (l, r) \in HR(\mathcal{M}, S_j) \\ \exists (l', r') \in MH(M_i, S_j) : l \leq l' < r' \leq r), \quad (7.1)$$

where $MH(M_i, S_j)$ is the set of motif hits of motif M_i on sequence S_j and $HR(\mathcal{M}, S_j)$ is the set of hit regions of the whole motif set $\mathcal{M}$ on sequence S_j.

In other words, if in a particular genomic sequence a particular motif is within a hit region of the motif set, this motif's variable for that sequence must be 1. Compared to itemset mining, $\widetilde{seqM}_{ij}$ can be thought of as a binary matrix in traditional itemset mining; however, here this matrix is dynamically defined by means of the proximity constraint. Observe that $\widetilde{seqM}_{ij} = 1$ will hold for all motifs M_i in the motif set $\mathcal{M}$ on all sequences S_j that are in $\varphi(\mathcal{M}, \mathcal{S})$. Of course, motifs not in $\mathcal{M}$ can also have hits in the proximity of a region in $HR(\mathcal{M}, S_j)$ on a sequence S_j, and hence have $\widetilde{seqM}_{ij} = 1$ for that motif and sequence.

Constraint (7.1) is a specialized constraint for our problem for which no efficient propagator is available in existing CP systems. Consequently we need to propose a new propagator, which we present later in this section.

Coverage Constraint Using the variables defined by the proximity constraint, we can define the sequence variables $\widetilde{S}$ as follows:

$$\forall j : \widetilde{S}_j = 1 \Leftrightarrow \sum_i \widetilde{M}_i(1 - \widetilde{seqM}_{ij}) = 0.$$

In other words, a sequence is covered if each motif is either not in the motif set, or has on this sequence a hit in a hit region of the motif set. This constraint is logically equivalent with the coverage constraint in itemset mining (Equation (3.2) on page 40). The main difference is that the constraint is defined over the dynamically calculated binary variables $\widetilde{seqM}_{ij}$, instead of a static data matrix. This binary constraint is available in most CP systems.

Frequency Constraint The constraint that imposes a minimum number of sequences to contain the motif set can now straightforwardly be formalized as:

$$\sum_j \widetilde{S}_j \geq \theta.$$

This is similar to the frequency constraint in itemset mining.

Redundancy Constraint As we pointed out earlier, exhaustive search is likely to consider a large number of solutions, some of which can be considered redundant with respect to each other. This is partly due to the density of the data (data typically consists of multiple binding sites for most motif and sequence combination). For instance, if a motif set consisting of 5 motifs $\{a, b, c, d, e\}$ meets the proximity and frequency constraints, then any of its

subsets $\{a, b, c, d\}, \{a, b, c, e\}, \ldots, \{b, c, e\}, \ldots, \{e\}$ will also contribute to CRMs that meet the same constraints and hence will be reported as a solution. Many of these subsets contribute to CRMs that occur in exactly the same sequences and often contain exactly the same binding regions as those identified for the larger superset. Tests on small datasets indicated that up to 80% of the solutions, and hence computation time, is spent on enumerating redundant solutions. To avoid these solutions, we imposed that a solution has to be maximally specific given the sequences that it covers. More formally, we require that

$$\forall i : \widetilde{M}_i = 1 \Leftrightarrow \sum_j \widetilde{S}_j (1 - \widetilde{seqM}_{ij}) = 0.$$

In other words, a motif has to be in the motif set if it has, for every covered sequence, a hit in a hit region of the motif set. Note that this constraint is logically equivalent to the closedness constraint for traditional closed itemset mining (Equation (3.7) on page 48). The difference is again that the $\widetilde{seqM}_{ij}$ variables are not given, but dynamically calculated by the proximity constraint.

Consequently, certain types of propagation which are always possible in traditional closed itemset mining are only possible when the domains of the variables $\widetilde{seqM}_{ij}$ are fixed. For instance, whereas in traditional closed itemset mining, if two items (motifs) always occur in the same sequences, we can always add them together, in our case, the proximity constraint may disallow this; the hits of the motifs may be too far from each other for the motifs to be added together in a set. This makes the adaption of traditional itemset mining algorithms difficult. In contrast, in CP we implemented a propagator for the proximity constraint. The principles of CP ensure that this constraint can be combined with others, and that the search will still be performed correctly and efficiently.

Propagation of the Constraints

The frequency, coverage and redundancy constraints are variants of the ones used in constraint programming for itemset mining and have been covered in Chapter 3. Here we focus on the propagation of the proximity constraint (7.1), which is an essential component of our proposed approach. Its goal is to propagate changes of the motif variables to the $\widetilde{seqM}_{ij}$ variables. Initially, the domains of the $\widetilde{seqM}_{ij}$ and $\widetilde{M}_i$ variables are $\{0, 1\}$. The propagator does the following:

Remove 1 The value 1 in the domain of $\widetilde{seqM}_{ij}$ indicates that at this point of the search it is still possible for motif i to appear in the proximity of the final CRM. Value 1 is removed when it does not hold that:

$$\exists (l, r) \in HR(\mathcal{M}, S_j), \exists (l', r') \in MH(M_i, S_j) : l \leq l' < r' \leq r,$$

where $\mathcal{M} = \{M_j \mid D(\widetilde{M}_j) = \{1\}\}$ represents the motifs that have been fixed in the search. In words, when a motif does not have a hit in the proximity of motifs already fixed, we remove it from consideration, as only motifs close enough to other motifs can be part of a potential CRM.

Remove 0 The value 0 in the domain of $\widetilde{seqM}_{ij}$ represents the possibility that there is a solution in which no hit of motif i in sequence j is part of a CRM. However, if the largest motif set still reachable actually has a hit region in sequence j fulfilling the proximity constraint, and motif i is within this region, we may conclude that motif i will always be part of the CRM. In other words, we remove 0 if:

$$\exists (l, r) \in HR(\mathcal{M}, S_j), \exists (l', r') \in MH(M_i, S_j) : l \leq l' < r' \leq r),$$

where $\mathcal{M} = \{M_j \mid 1 \in D(\widetilde{M}_j)\}$.

The removal of zeros is in particular important as variables fixed to 1 ($D(\widetilde{seqM}_{ij}) = \{1\}$) can be exploited to propagate other constraints; when a sufficient number of motifs is fixed, it may be concluded that certain other motifs will always be part of the CRM.

7.4 Experiments

In this section we evaluate the effectiveness of our approach by answering the following three research questions:

- Q1. How well does our algorithm improve the results of the screening phase?
- Q2. What is the effect of the proximity constraint on the quality of the result?
- Q3. How does our method compare to state-of-the-art methods?

To answer these questions we use data constructed by Xie et al. 2008 [Xie et al. 2008]. The data consists of 22 genomic sequences of the mouse genome, each

1000 base pairs in length. In the first 20 sequences, transcription factors Oct4, Sox2 and FoxD3 are each inserted 3 times, in a region of at most 164bp. The inserted nucleotides are sampled from the respective TRANSFAC PWMs. The last two sequences have no inserted transcription factors[2].

All the methods under evaluation start from a given set of motif models. We use 516 transcription factors (TFs) from the TRANSFAC database [Matys et al. 2003], obtained after filtering all 584 vertebrate TFs using the motif similarity checking tool MotifComparison [Coessens et al. 2003] with a threshold of 0.1.

We measure the quality of a method's prediction by comparing the best solution it finds with the inserted modules. In all methods, a solution consists of at most one predicted set of motifs for every sequence, with for every motif a number of hits at which it is predicted to bind. Motif detection tools that search over multiple sequences predict one set of motifs with hits in a number of those sequences. Often a binding region for the whole motif set is returned instead of hits for each motif separately.

As in [Klepper et al. 2008], we evaluate solutions both at the motif level and at the nucleotide level. At the motif level, a predicted motif for a sequence is a true positive (TP) if that motif was indeed inserted in that sequence, otherwise it is a false positive (FP). If a motif was not predicted, but was inserted in that sequence, it is counted as a false negative (FN), otherwise as a true negative (TN). As the motif-level evaluation does not take the predicted binding sites into account, we also evaluate a solution at the nucleotide level: for every nucleotide we verify whether it was predicted to be part of the CRM and whether it should have been predicted or not, again resulting in TP, FP, FN, and TN counts. These counts are aggregated over all sequences to obtain the total counts of this solution. Ideally, a solution scores good at both the motif and nucleotide level.

In the following experiments we use the Clover tool [Frith et al. 2004] in the screening phase. The default thresholds are used unless mentioned otherwise and no randomization options were used. In the mining phase we use a proximity threshold of 165 bp and require a minimum frequency of 60%. There is no limit on the maximum number of motifs that can be selected. In the ranking phase we use 2000 randomly selected non-repeating intergenic sequences from the Mouse genome. The best ranked CRM is the solution evaluated in the experiments.

[2]Datasets and CPmodule software (based on the Gecode Constraint Programming system) available at `http://dtai.cs.kuleuven.be/CP4IM/CPmodule/`

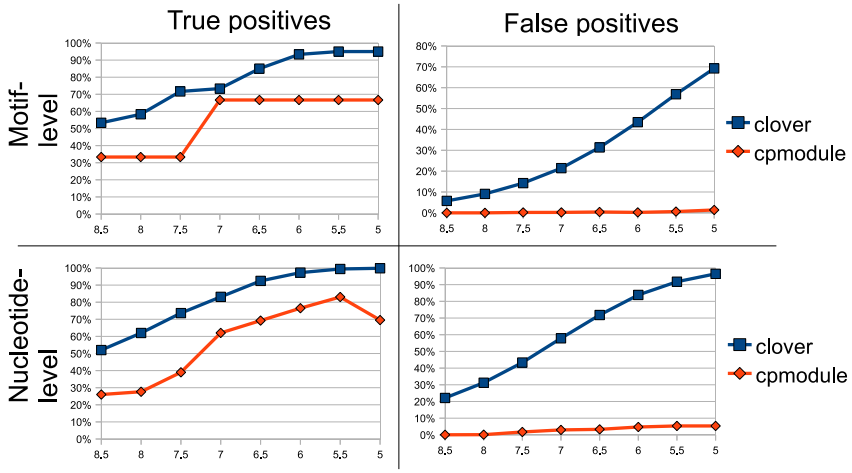


Figure 7.3: Comparison between the single motif scanning tool (phase 1) and our method (phase 3); on the x axis is the raw threshold provided to the single motif scanning tool.

7.4.1 Improvements over the Screening Phase

To answer Q1, we compare the quality of the motifs and hits predicted by the screening tool in phase 1 with the best module predicted after phase 3. In the screening phase we vary the threshold on the hit score. A lower score results in more hits of lower quality. Figure 7.3 lists the relative number of true and false positives at both the motif level (top row) and nucleotide level (bottom row) for different hit score thresholds. At the motif level, when decreasing the hit score, the screening tool (blue line) predicts more true motifs correctly, however the number of falsely predicted motifs increases at a far larger rate. This confirms earlier findings [Tompa et al. 2005] that single motif detection tools have a large number of false positives. In contrast, our method (red line) finds many true positives while including only a fraction of the false positives. Even when the screening tool returns more than 50% false positives at the motif level, our algorithm selects at most 1% false positives. The situation is similar at the nucleotide level: our algorithm returns a solution that has many true positives, yet only a fraction of the false positives. Note that with a hit score of 5 the performance of our algorithm decreases. This relates to the fact that at this threshold, the density of hits gets so large that an increasing number of motifs will have a hit in each other's proximity matching the algorithm's criterion. This leads to a considerable decrease in predictive and computational performance.

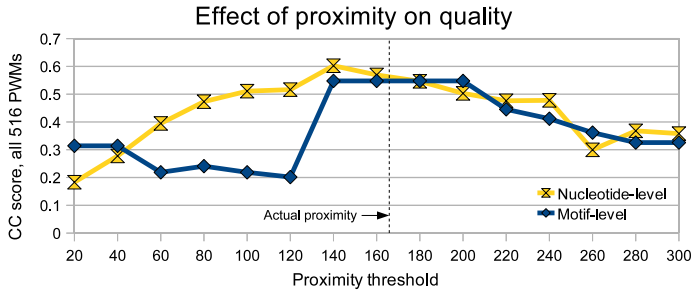


Figure 7.4: Effect of the proximity constraint on quality.

7.4.2 Effect of the Proximity Constraint

To answer Q2, we run our method using different proximity threshold values. Figure 7.4 shows the motif level and nucleotide level Correlation Coefficient (CC) calculated as in [Klepper et al. 2008], also known as Matthews Correlation Coefficient:

$$CC = \frac{TP * TN - FN * FP}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}} \tag{7.2}$$

The higher the score, the better the solution correlates with the true CRMs. The known maximum proximity of 164bp is indicated by a vertical line, we ran our algorithm with an increasingly large proximity threshold. The binding region predicted by our algorithm is constrained to be at most the size of the proximity threshold, hence at a low threshold the nucleotide level score is low and modules with at most one true motif are found. Using thresholds between 140 and 200bp our method achieves its highest motif level score, but the method is not sensitive to the exact setting of the parameter within this range. The nucleotide level score is more sensitive to the proximity threshold as a larger threshold results in larger predicted binding regions and hence more false positively predicted nucleotides.

7.4.3 Comparison with Other Algorithms

We compared the performance of CPModule with four other methods, namely, Cister [Frith et al. 2001], Cluster-Buster [Frith et al. 2003], ModuleSearcher [Aerts et al. 2004] and Compo [Sandve et al. 2008]. We used the parameter values listed in table 7.1 and default values otherwise.

Algorithm	Parameter	Setting
Cister	avg. distance between motifs	20
Cluster-Buster	gap	20
	residue abundance range	1000
ModuleSearcher	search algorithm	genetic algorithm
	average number of motifs	6 (as Cister)
	maximum CRM size	165
	multiple copies of TF	no
	incomplete CRM penalty	no
	GA iterations	300
Compo	forbid overlaps	yes
	number of motifs	between 1 and 20
	distance window	165
	tp-factors	2, 3 and 4
	background sequences	same as CPModule

Table 7.1: Non-default settings for alternative algorithms

	Cister	Cluster-Buster	ModuleSearcher	Compo	Cpmodule
mCC	0.16	0.05	/	-	**0.57**
nCC	0.23	0.23	/	-	**0.55**

/ indicates termination by lack of memory,
- indicates that the algorithm was still running after 2 days.

Table 7.2: Motif and nucleotide level CC scores for different algorithms using all 516 TRANSFAC PWMs

Table 7.2 shows the motif and nucleotide level CC scores for the different algorithms, when using all 516 transfac PWMs. Cister and Cluster-Buster are able to find a solution, albeit of mediocre quality. The reason is that they operate on each sequence individually, which results in a rather large number of different motif predictions per sequence, most of which are false positives. ModuleSearcher was unable to finish because of memory problems, even when being allocated 2GB of ram. Compo was still running after 2 days and without any results returned. Compo also uses a strategy based on itemset mining, but does not address the problem of redundancy, which is inevitable when dealing with large numbers of motifs and hits.

To better compare the algorithms we ran them with an increasing number of PWMs. We create different PWM sets by starting from the 3 PWMs of the inserted TFs, and sampling a number of additional PWMs from the set of 513 remaining PWMs. We independently sample 10 sets for every sample size. Figure 7.5 shows the correlation coefficient scores of the different

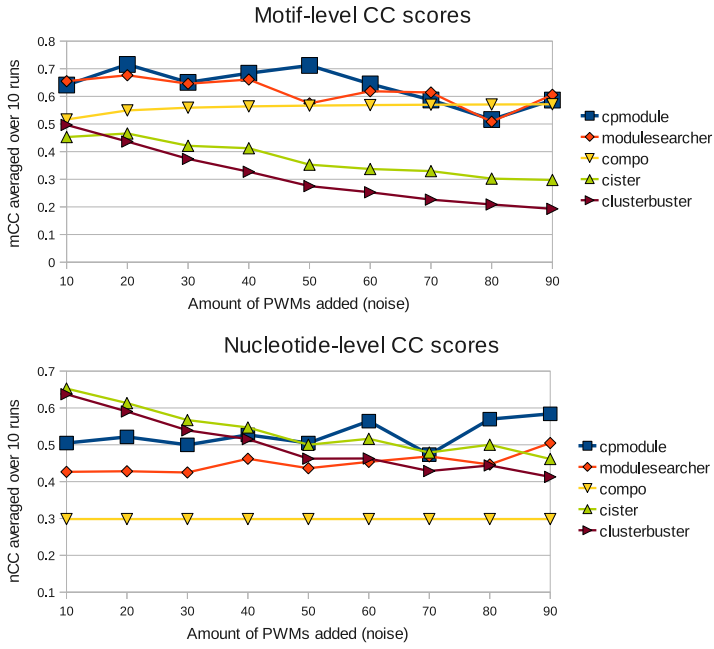


Figure 7.5: Comparing motif and nucleotide level correlation coefficients of several algorithms.

algorithms when run with an increasing number of sampled PWMs. At the motif level, CPModule often preforms best, closely followed by ModuleSearcher. On the nucleotide level, however, CPModule performs significantly better than ModuleSearcher. Compo has a mostly constant performance at the motif level, however its nucleotide level score is rather low. Independent of the PWM set given, it returns just one of the inserted motifs on each of the sequences. We suspect that its screening method was too stringent, but found no way to change this behaviour. Cister and Cluster-Buster have low motif level scores, yet reasonably high nucleotide level scores, especially when using fewer PWMs. This shows that they are good tools for finding small CRMs on single sequences, but are not competitive to tools that operate on multiple sequences.

CPmodule on the other hand is able to effectively find CRMs over multiple sequences, even when given a large amount of PWMs. It is hence best in handling the combinatorial nature of CRM detection in co-regulated genes.

7.5 Conclusions

In this chapter we presented how the problem of finding motif sets in co-regulated genomic sequences can be formulated in our constraint programming based itemset mining framework. This has several advantages: existing constraints and their efficient implementation can be used out-of-the-box; the proximity constraint can easily be added to the system; existing itemset mining principles such as closedness can be applied readily. We demonstrated the usefulness of this approach by creating the CPModule system. This system can be used in a 3-phase method in which first a set of putative binding sites is computed by a motif detection tool. The CPModule system takes these binding sites as input and finds all sets of motifs that appear in each other's proximity in a sufficient number of sequences. This set of solutions is ranked according to p-value, calculated against a large set of background sequences. The method was evaluated experimentally. When we compare the output of the screening tool with the output of our method, we observe a significant reduction in false positives without significantly affecting true positives. The proximity constraint was shown to have a positive influence on predictive performance, while the sensitivity to this constraint was not too large. Compared to other tools, the predictive performance is usually competitive or better.

In future work we plan to further exploit the extensibility of our framework to deal with additional constraints: for instance, disallowing overlaps between motif hits or considering different strategies for avoiding statistical redundancies. Furthermore, we plan to apply our methodology on different types of data, such as ChIP-Seq data.

Chapter 8

Conclusions and Future Work

This chapter concludes the thesis and provides a discussion of open problems and opportunities for future work.

8.1 Summary and conclusions

Our motivation to develop declarative pattern mining methods stems from the generally accepted use of constraints to guide the knowledge discovery process. The popularity of constraints had led to the creation of constraint-based mining frameworks, using anti-monotonicity as guiding principle. However, existing frameworks are restricted in the constraints they support, for example they can neither handle arbitrary combinations of constraints, nor condensed representations. A truly general approach has been missing.

The main contribution of this thesis is that we use constraint programming as a general declarative framework for constraint-based mining. The key point of constraint programming is that modeling a problem is decoupled from solving it. The resulting flexibility comes from the fact that every constraint is independent from the other constraints; on the modeling side, arbitrary combinations of constraints can be specified while on the solver side highly effective propagators can be devised for the individual constraints. We have successfully applied the constraint programming methodology on a range of data mining problems.

In the introduction we identified three research questions. We now review these questions and the answers provided in this thesis.

Q1 *What can constraint programming offer to data mining?*
The main idea that constraint programming can offer to data mining is its declarative approach. The use of a high-level declarative language allows us to formulate many problems in the same framework. This is uncommon in data mining, where procedural approaches are the norm. Existing constraint-based mining systems do offer support for multiple constraints, but combining these constraints in unforeseen ways or adding complex constraints requires changing the underlying algorithms. Constraint programming achieves a greater flexibility because every constraint can independently reduce the search tree. Hence, new constraints can easily be added and constraints can be freely combined and reused without modifying the underlying algorithms. This makes constraint programming an ideal framework for testing the effectiveness of individual constraints and the prototyping of alternative problem formulations. Also, in Chapter 3 we showed that the consistency principles of constraint programming can improve the pruning of complex constraints such as correlation constraints. Using an out-of-the-box constraint solver with this improved propagator outperformed existing specialised algorithms. Additionally, in Chapter 4, we showed how the constraint programming framework can act as a basic framework for investigating the computational complexity of certain mining tasks. For closed itemset mining and a related graph mining example we proved that the solutions can be enumerated with polynomial delay.

Q2 *To what range of data mining problems can constraint programming be applied?*
The declarative approach of constraint programming allows applying it on a wide range of data mining problems. What is necessary is to formulate a problem as a constraint satisfaction or constraint optimisation problem. We have introduced such formulations for many pattern mining problems, including constraint-based mining, discriminative itemset mining (Chapter 3) and the related problem of k-pattern set mining, where the goal is to find a compact set of k patterns that together satisfy the constraints (Chapter 5). In doing so, we have introduced the *constraint programming for itemset mining* framework which is the most general constraint-based mining system to date.
Additionally, in Chapter 7 we applied our framework to a challenging bioinformatics problem. We presented a solution method for *cis*-regulatory module detection, a problem regarding the identification of key transcription factors in gene regulation. This was a real-life test of the applicability of constraint programming on novel data mining problems, as solving this problem required the use of domain-specific knowledge; standard mining techniques demanded pre-processing the data in ways that resulted in erroneous solutions.

We have mostly used solvers that employ exhaustive search, which is not always

feasible for data mining problems. In Chapter 6 we provided initial evidence that constraint programming is not limited to tasks for which exhaustive search is feasible. We evaluated different search strategies for pattern set mining, both heuristic and complete as well as in one step or in a two step procedure. Constraint programming was successfully used in each of the strategies, making a principled comparison possible.

Q3 *What are the limitations of using constraint programming for data mining?*
The main limitation of constraint programming involves the efficiency of the solving. Although we often achieved acceptable performance with an out-of-the-box constraint solver, such a solver cannot be expected to perform as well as a highly-optimised algorithm for a specific task. In Chapter 4 we compared in detail the difference in performance between a standard constraint programming solver and specialised mining algorithms. We showed that the difference is not fundamentally due to the principles of CP, but rather due to the representation of the variables and the data in the CP solver. In a CP solver, the data is encoded into the constraints, and the maintenance and unnecessary activation of these constraints introduces extra overhead. We showed that such an overhead can be avoided by introducing Boolean vectors as variable type, which makes it possible to use multiple data representations and efficient matrix calculations. We implemented this in a hybrid solver that employs the generic principles of CP while having similar performance as specialised miners.
In Chapter 5 we investigated the use of constraint programming and exhaustive search for pattern set mining tasks. Depending on the task, and more specifically on the constraints used to formulate the task, the scalability of this approach was limited. Good results were obtained for problems that could be formulated using constraints known to propagate well during search. In other cases, the exhaustive search was prohibitively slow. This calls for more research into effective propagators for certain constraints, and the use of more heuristic search techniques.

In general, our results show that constraint programming provides a promising perspective on, and approach to, data mining. Its declarative nature offers reuse of constraints, ease of experimentation and iterative refinement of problem specifications. Its constraint-based search offers a more flexible framework than existing specialised systems. This thesis is only a first step in using constraint programming for data mining. It has sparked a number of interesting studies, such as the applicability of other declarative solving techniques like knowledge compilation techniques [Cambazard et al. 2010], Answer Set Programming [Järvisalo 2011] and SAT solving [Métivier et al. 2011], and the extension of our approach to n-ary patterns and rules [Khiari et al. 2010] and sequence mining

[Coquery et al. 2011]. Many other open questions remain that deserve further study.

8.2 Discussion and Future work

We end this thesis with a short discussion of five promising directions for future research, namely: 1) CP solvers specialised for data mining; 2) complex constraints; 3) beyond exhaustive search; 4) other data mining tasks; and a general open question to conclude, 5) can constraint programming serve as unifying platform for data mining?

CP solvers specialised for data mining. In our applications we have extensively used the standard Gecode constraint programming solver, extended with additional constraints where needed. This solver was not built with data mining applications in mind, and hence its scalability towards large datasets was limited at times. This should be addressed before constraint programming can be used as an industrial data mining tool.

The specialised hybrid solver developed in Chapter 4 is an initial step in this direction. This work can be extended in two ways. The first is to extend the hybrid solver with additional constraints until it is on par with other constraint solvers. This would allow a comparison between the hybrid solver and CP solvers, as well as the application of the hybrid solver to more problems. The second option is to port back the relevant features to an existing constraint solver. For example, the Boolean vector variable type and matrix constraints could be plugged into a standard solver. This could benefit existing solvers while possibly leading to further insights in how to improve such solvers.
A more radical approach to the second option could also study how data structures specifically designed for data mining, such as FP-trees [Han et al. 2000], could be integrated in a hybrid constraint solver.

Another limitation of our work is that we did not study the use of other CP solvers in detail. When comparing CP solvers, we observed runtime differences that were often caused by a difference in the host language. A detailed study of the difference in performance of solvers using copying or trailing [Schulte 1999], or even lazy clause generation [Ohrimenko et al. 2007] could lead to novel insights.

Recently, a number of other declarative methods have been proposed for solving itemset mining problems: Cambazard et al. [2010] propose a knowledge compilation process involving binary decision diagram-like circuits, Järvisalo

[2011] proposes the use of answer set programming and Métivier et al. [2011] propose a compilation to SAT.

The use of SAT solving seems natural given our extensive use of constraints over Booleans. However, constraints such as the frequency constraint require integer arithmetic, which has to be encoded into SAT as well. For large datasets, encoding large integer numbers can become costly. Perhaps pseudo-boolean solvers provide a good alternative in this case. When dealing with optimisation problems, another alternative could be the use of Integer Linear Programming (ILP) solvers. Many of our constraints are linear sums over boolean variables. However, the use of ILP methods is complicated by our extensive use of reified constraints such as $B \leftrightarrow \sum V \leqslant \theta$. Making such constraints linear requires the use of Big-M reformulations [Winston and Venkataramanan 2003]. Unfortunately such formulations are typically difficult to solve; we leave it as future work to investigate this in more detail.

In general, the advantages and disadvantages of different solver technology deserve further study; not just in terms of efficiency but also in terms of flexibility and generality towards new constraints.

Complex constraints. Although we showed for a large number of diverse constraints how to formulate them in a constraint programming framework, many other constraints and tasks exist. Some of these constraints do not fit very well in a constraint programming framework, as is the case for a number of condensed representations [Boulicaut et al. 2000; Calders and Goethals 2007]. For example, to determine whether an itemset is a non-derivable one [Calders and Goethals 2007], the frequency of all its subsets must be known. This is possible using depth-first search, but requires a strict predefined ordering of the variables during search [Calders and Goethals 2005]. Other condensed representation constraints also require that the frequency of some of its subsets is known.

For other complex constraints it is not (yet) clear how to effectively use them to reduce the search space. A striking example is a constraint on the area of an entire pattern set, where patterns are interpreted as tiles in a binary database (Section 5.3.1). The natural formalization of this constraint involves $|\mathcal{I}| * |\mathcal{S}|$ variables, where the size of $\mathcal{S}$ can grow very large. It is an open question whether good propagators can be devised for such constraints.

An aspect that is not often studied in data mining is whether constraints can be decomposed into smaller constraints. We have shown in Chapter 5 how some global constraints can be decomposed into pairwise constraints. Perhaps the idea of decomposing constraints can also be applied to other mining problems?

Beyond exhaustive search. We employed depth-first search in our constraint programming framework as it mimics the search of state-of-the-art itemset mining algorithms. However, constraint solvers support other types of search as well, including limited discrepancy search [Harvey and Ginsberg 1995] and search with restarts [Gomes et al. 1998]. Advanced solvers also exist in which heuristic search, e.g. local search, can be employed [Van Hentenryck and Michel 2009]. This is a promising approach for problems that cannot be solved with exhaustive search. For example in Chapter 6 we compared a number of heuristic pattern set mining strategies, but these were implemented as wrappers using ad-hoc scripts. We have recently set a first step in the direction of employing declarative heuristic search methods on different pattern set mining problems [Guns et al. 2011d].

We believe the use of declarative methods in which both exhaustive and heuristic search can be performed would increase the applicability of constraint programming to data mining problems of any size.

Other data mining tasks. In this work we focused mostly on itemset mining tasks. Other types of pattern mining have been studied, where patterns can be sequences, trees or graphs. The use of constraint programming for such structured types of patterns has not been studied extensively yet. In contrast to items in a set, elements in a sequence or nodes in a graph can be recurring. Hence, one major challenge is the identification of suitable upper-bounds on the size of pattern to be considered. This makes it harder to design good bounds-consistent propagators, or to develop efficient look-ahead constraints. For graphs there is the additional problem that one has to repeatedly check whether a pattern covers an example in the data. This would require the repeated solving of the subgraph isomorphisms problem, which is known to be NP complete [Garey and Johnson 1990].

Many other data mining tasks exist, such as text mining, link prediction, matrix factorization and clustering. Most notable is the recent work of Davidson et al. [2010] and Gilpin and Davidson [2011] on using constraint satisfaction techniques for constrained clustering tasks. It is an interesting open question as to whether in general, such problems fit in a constraint programming framework too.

Additionally, the use of SAT solvers [Cussens 2008; Gilpin and Davidson 2011; Métivier et al. 2011] and mathematical programming [Chang et al. 2008; Saigo et al. 2009; Humrich et al. 2011] in data mining is increasingly common. An inherent question is whether CP solvers can improve these works in cases where the current approaches are too restrictive, for example, when constraints other than clauses or linear constraints are needed. The generality of CP compared

to SAT and ILP does come at a computational cost, but current developments aimed at combining SAT and ILP with CP may lead to more opportunities for constraint programming in data mining as well.

Can constraint programming serve as unifying platform for data mining? There is still no generally accepted theory or method for data mining. Perhaps the declarative methodology of constraint programming could act as a unifying platform for data mining? Such a platform would require a common language in which data mining problems could be specified [De Raedt and Nijssen 2011]. Such a language would be expected to contain data mining specific concepts such as pattern language, (structured) data, distances, coverage of a pattern and so forth. Currently many search strategies are used in data mining hence the framework would need multiple solvers that support exhaustive search, greedy search, convex optimisation, sampling, ... Most importantly, the system should allow one to combine problem and search specifications and make it easy to adapt them. This could facilitate a truly iterative approach to data mining, which is an essential step in the knowledge discovery process.

Bibliography

Aerts, S., Van Loo, P., Moreau, Y., and De Moor, B. (2003). Computational detection of cis -regulatory modules. *Bioinformatics*, 19(Supp 1):ii5–14. pages

Aerts, S., Van Loo, P., Moreau, Y., and De Moor, B. (2004). A genetic algorithm for the detection of new cis-regulatory modules in sets of coregulated genes. *Bioinformatics*, 20(12):1974–76. pages

Afrati, F., Gionis, A., and Mannila, H. (2004). Approximating a collection of frequent sets. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 12–19. ACM. pages

Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216. ACM Press. pages

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1996). Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press. pages

Alves, R., Rodriguez-Baena, D. S., and Aguilar-Ruiz, J. S. (2010). Gene association analysis: a survey of frequent pattern mining from gene expression data. *Briefings in Bioinformatics*, 11(2):210–224. pages

Apt, K. R. and Wallace, M. (2007). *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA. pages

Arimura, H. and Uno, T. (2009). Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proceedings of the SIAM International Conference on Data Mining*, pages 1087–1098. SIAM. pages

Bay, S. D. and Pazzani, M. J. (1999). Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 302–306. ACM Press. pages

Bayardo, R. (1998). Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93. ACM. pages

Bayardo, R., Agrawal, R., and Gunopulos, D. (2000). Constraint-based rule mining in large, dense databases. *Data Min. Knowl. Discov.*, 4(2/3):217–240. pages

Becquet, C., Blachon, S., Jeudy, B., Boulicaut, J. F., and Gandrillon, O. (2002). Strong-association-rule mining for large-scale gene-expression data analysis: a case study on human SAGE data. *Genome Biology*, 3(12). pages

Beldiceanu, N., Carlsson, M., Demassey, S., and Petit, T. (2007). Global constraint catalogue: Past, present and future. *Constraints*, 12:21–62. pages

Benhamou, F., Jussien, N., and O'Sullivan, B. (2007). *Trends in Constraint Programming*. ISTE, London, UK. pages

Bessiere, C. (2006). Constraint propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, pages 29–84. Elsevier. pages

Blockeel, H., Calders, T., Fromont, É., Goethals, B., Prado, A., and Robardet, C. (2008). An inductive database prototype based on virtual mining views. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1061–1064. ACM. pages

Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A., and Robardet, C. (2011). A practical comparative study of data mining query languages. In Dzeroski, S., Goethals, B., and Panov, P., editors, *Inductive Databases and Constraint-Based Data Mining*, pages 59–77. Springer. pages

Boley, M., Horváth, T., Poigné, A., and Wrobel, S. (2007). Efficient closed pattern mining in strongly accessible set systems. In *Knowledge Discovery in Databases, European Conference (PKDD 2007)*, volume 4702 of *Lecture Notes in Computer Science*, pages 382–389. Springer. pages

Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., and Trasarti, R. (2009). A constraint-based querying system for exploratory pattern discovery. *Inf. Syst.*, 34:3–27. pages

Bonchi, F. and Goethals, B. (2004). Fp-bonsai: The art of growing and pruning small fp-trees. In *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 155–160. Springer. pages

Bonchi, F. and Lucchese, C. (2007). Extending the state-of-the-art of constraint-based pattern discovery. *Data Knowl. Eng.*, 60(2):377–399. pages

Borgelt, C. (2003). Efficient implementations of Apriori and Eclat. In *Workshop of Frequent Item Set Mining Implementations (FIMI)*. pages

Boulicaut, J.-F., Bykowski, A., and Rigotti, C. (2000). Approximation of frequency queries by means of free-sets. In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Computer Science*, pages 24–43. Springer Berlin / Heidelberg. pages

Bringmann, B., Nijssen, S., Tatti, N., Vreeken, J., and Zimmermann, A. (2010). Mining sets of patterns. In *Tutorial at ECMLPKDD 2010*. pages

Bringmann, B., Nijssen, S., and Zimmermann, A. (2009). Pattern-based classification: A unifying perspective. In *LeGo, 'From Local Patterns to Global Models', Second ECML PKDD Workshop*. pages

Bringmann, B. and Zimmermann, A. (2005). Tree^2 - decision trees for tree structured data. In *Knowledge Discovery in Databases, European Conference (PKDD 2005)*, volume 3721 of *Lecture Notes in Computer Science*, pages 46–58. Springer. pages

Bringmann, B. and Zimmermann, A. (2007). The chosen few: On identifying valuable patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 63–72. IEEE Computer Society. pages

Bucila, C., Gehrke, J., Kifer, D., and White, W. M. (2003). Dualminer: A dual-pruning algorithm for itemsets with constraints. *Data Min. Knowl. Discov.*, 7(3):241–272. pages

Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., and Yiu, T. (2005). MAFIA: A maximal frequent itemset algorithm. *IEEE Trans. Knowl. Data Eng.*, 17(11):1490–1504. pages

Calders, T. and Goethals, B. (2005). Depth-first non-derivable itemset mining. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, pages 250–261. SIAM. pages

Calders, T. and Goethals, B. (2007). Non-derivable itemset mining. *Data Min. Knowl. Discov.*, 14(1):171–206. pages

Cambazard, H., Hadzic, T., and O'Sullivan, B. (2010). Knowledge compilation for itemset mining. In *19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 1109–1110. IOS Press. pages

Carlsson, M., Ottosson, G., and Carlson, B. (1997). An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programs*, volume 1292 of *Lecture Notes in Computer Science*, pages 191–206. Springer. pages

Chang, M.-W., Ratinov, L.-A., Rizzolo, N., and Roth, D. (2008). Learning and inference with constraints. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1513–1518. AAAI Press. pages

Cheng, H., Yan, X., Han, J., and Hsu, C.-W. (2007). Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 716–725. IEEE. pages

Cheng, H., Yan, X., Han, J., and Yu, P. (2008). Direct discriminative pattern mining for effective classification. In *Proceedings of the 24th International Conference on Data Engineering*, pages 169 –178. IEEE. pages

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283. pages

Coessens, B., Thijs, G., Aerts, S., Marchal, K., De Smet, F., Engelen, K., Glenisson, P., Moreau, Y., Mathys, J., and De Moor, B. (2003). INCLUSive: a web portal and service registry for microarray and regulatory sequence analysis. *Nucleic Acids Research*, 31(13):3468–3470. pages

Coquery, E., Jabbour, S., and Saïs, L. (2011). A constraint programming approach for enumerating motifs in a sequence. In *Proceedings of the International ICDM Workshop on Declarative Pattern Mining*. IEEE Computer Society. To appear. pages

Cussens, J. (2008). Bayesian network learning by compiling to weighted max-sat. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 105–112. AUAI Press. pages

Davidson, I., Ravi, S. S., and Shamis, L. (2010). A sat-based framework for efficient constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining*, pages 94–105. SIAM. pages

De Raedt, L., Guns, T., and Nijssen, S. (2008). Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-08)*, pages 204–212. ACM. pages

De Raedt, L., Guns, T., and Nijssen, S. (2010). Constraint programming for data mining and machine learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1671–1675. AAAI Press. pages

De Raedt, L. and Nijssen, S. (2011). Towards programming languages for machine learning and data mining (extended abstract). In *Foundations of Intelligent Systems - 19th International Symposium*, volume 6804 of *Lecture Notes in Computer Science*, pages 25–32. Springer. pages

De Raedt, L. and Zimmermann, A. (2007). Constraint-based pattern set mining. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, pages 1–12. SIAM. pages

Diaz, D. and Codognet, P. (2001). Design and implementation of the gnu prolog system. *Journal of Functional and Logic Programming*, 2001(6). pages

Dong, G. and Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 43–52. ACM Press. pages

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 *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–238. ACM. pages

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172. pages

Frank, A. and Asuncion, A. (2010). UCI machine learning repository. Available from `http://archive.ics.uci.edu/ml`. pages

Frank, E. and Witten, I. H. (1998). Using a permutation test for attribute selection in decision trees. In *Proc. 15th International Conf. on Machine Learning*, pages 152–160. pages

Frisch, A. M., Harvey, W., Jefferson, C., Hernández, B. M., and Miguel, I. (2008). Essence : A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306. pages

Frith, M. C., Fu, Y., Yu, L., Chen, J.-F., Hansen, U., and Weng, Z. (2004). Detection of functional DNA motifs via statistical over-representation. *Nucleic Acids Res*, 32(4):1372–81. pages

Frith, M. C., Hansen, U., and Weng, Z. (2001). Detection of cis-element clusters in higher eukaryotic DNA. *Bioinformatics*, 17(10):878–889. pages

Frith, M. C., Li, M. C., and Weng, Z. (2003). Cluster-buster: finding dense clusters of motifs in DNA sequences. *Nucleic Acids Research*, 31(13):3666–3668. pages

Fujibuchi, W., Kim, H., Okada, Y., Taniguchi, T., and Sone, H. (2009). High-performance gene expression module analysis tool and its application to chemical toxicity data. *Methods Mol. Biol.*, 577:55–65. pages

Fürnkranz, J. and Flach, P. A. (2005). ROC 'n' rule learning – towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77. pages

Gallo, A., De Bie, T., and Cristianini, N. (2007). MINI: Mining informative non-redundant itemsets. In *Knowledge Discovery in Databases: European Conference (PKDD 2007)*, volume 4702 of *Lecture Notes in Computer Science*, pages 438–445. Springer. pages

Ganter, B., Stumme, G., and Wille, R., editors (2005). *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer. pages

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA. pages

Garriga, G. C., Kralj, P., and Lavrac, N. (2008). Closed sets for labeled data. *Journal of Machine Learning Research*, 9:559–580. pages

Gecode Team (2010). Gecode: Generic constraint development environment. Available from `http://www.gecode.org`. pages

Geerts, F., Goethals, B., and Mielikäinen, T. (2004). Tiling databases. In *Discovery Science, 7th International Conference*, volume 3245 of *Lecture Notes in Computer Science*, pages 278–289. Springer. pages

Gent, I. P., Jefferson, C., and Miguel, I. (2006a). Minion: A fast scalable constraint solver. In *Proceeding of the 17th European Conference on Artificial Intelligence*, pages 98–102. IOS Press. pages

Gent, I. P., Jefferson, C., and Miguel, I. (2006b). Watched literals for constraint propagation in minion. In *Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 182–197. Springer. pages

Gervet, C. (1997). Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244. pages

Gilpin, S. and Davidson, I. (2011). Incorporating sat solvers into hierarchical clustering algorithms: an efficient and flexible approach. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1136–1144. ACM. pages

Goethals, B. and Zaki, M. J. (2004). Advances in frequent itemset mining implementations: report on FIMI'03. In *SIGKDD Explorations*, volume 6, pages 109–117. pages

Gomes, C. P., Selman, B., and Kautz, H. A. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence Conference (AAAI 98)*, pages 431–437. AAAI Press / The MIT Press. pages

Grosskreutz, H., Rüping, S., and Wrobel, S. (2008). Tight optimistic estimates for fast subgroup discovery. In *Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *Lecture Notes in Computer Science*, pages 440–456. Springer. pages

Guns, T., Nijssen, S., and De Raedt, L. (2010a). k-Pattern set mining under constraints. CW Reports CW596, Department of Computer Science, K.U.Leuven. pages

Guns, T., Nijssen, S., and De Raedt, L. (2011a). Evaluating pattern set mining strategies in a constraint programming framework. In *Advances in Knowledge Discovery and Data Mining, 15th Pacific-Asia Conference (PAKDD-11)*, volume 6635 of *Lecture Notes in Computer Science*, pages 382–394. Springer. pages

Guns, T., Nijssen, S., and De Raedt, L. (2011b). Itemset mining: A constraint programming perspective. *Artificial Intelligence*, 175(12-13):1951–1983. pages

Guns, T., Nijssen, S., and De Raedt, L. (2011c). k-pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, To Appear. Also as Technical Report CW596, Oct 2010. pages

Guns, T., Nijssen, S., Zimmermann, A., and De Raedt, L. (2011d). Declarative heuristic search for pattern set mining. In *Proceedings of the International ICDM Workshop on Declarative Pattern Mining*. IEEE Computer Society. To appear. pages

Guns, T., Sun, H., Marchal, K., and Nijssen, S. (2010b). Cis-regulatory module detection using constraint programming. In *Proceedings of the IEEE International Conference on Bioinformatics & Biomedicine (BIBM-10)*, pages 363–368. IEEE Computer Society. pages

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. pages

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press. pages

Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 607–615. pages

Holt, J. D. and Chung, S. M. (1999). Efficient mining of association rules in text databases. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management*, pages 234–242. ACM. pages

Huang, Y., Li, H., Hu, H., Yan, X., Waterman, M. S., Huang, H., and Zhou, X. J. (2007). Systematic discovery of functional modules and context-specific functional annotation of human genome. *Bioinformatics (Oxford, England)*, 23(13):i222–229. pages

Humrich, J., Gaertner, T., and Garriga, G. (2011). A Fixed Parameter Tractable Integer Program for Finding the Maximum Order Preserving Submatrix. In *Proceedings of the IEEE International Conference of Data Mining, ICDM 2011*, Vancouver, Canada. pages

ILOG S.A. (2003). ILOG Solver 6.0: Reference manual. Gentilly, France. pages

Imielinski, T. and Mannila, H. (1996). A database perspective on knowledge discovery. *Commun. ACM*, 39:58–64. pages

Imielinski, T. and Virmani, A. (1999). MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3:373–408. pages

Järvisalo, M. (2011). Itemset mining as a challenge application for answer set enumeration. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference*, volume 6645 of *Lecture Notes in Computer Science*, pages 304–310. Springer. pages

Kavsek, B., Lavrac, N., and Jovanoski, V. (2003). APRIORI-SD: Adapting association rule learning to subgroup discovery. In *Advances in Intelligent Data Analysis*, volume 2810 of *Lecture Notes in Computer Science*, pages 230–241. Springer. pages

Kearns, M. J. and Vazirani, U. V. (1994). *An introduction to computational learning theory.* MIT Press, Cambridge, MA, USA. pages

Khiari, M., Boizumault, P., and Crémilleux, B. (2010). Constraint programming for mining n-ary patterns. In *Principles and Practice of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer. pages

Klepper, K., Sandve, G., Abul, O., Johansen, J., and Drabløs, F. (2008). Assessment of composite motif discovery methods. *BMC Bioinformatics*, 9. pages

Knobbe, A. J. and Ho, E. K. Y. (2006). Pattern teams. In *Knowledge Discovery in Databases, European Conference (PKDD 2006)*, volume 4213 of *Lecture Notes in Computer Science*, pages 577–584. Springer. pages

Kuchcinski, K. and Szymanek, R. (2011). JaCoP Library User's Guide. Available from `http://www.jacop.eu/`. pages

Laburthe, F. (2000). CHOCO: implementing a CP kernel. In *Proceedings of TRICS: Techniques FoR Implementing Constraint programming Systems, a post conference workshop of CP 2000*, Technical Report TRA9/00, pages 71–85. School of Computing, National University of Singapore. pages

Lee, W., Stolfo, S. J., and Mok, K. W. (2000). Adaptive intrusion detection: A data mining approach. *Artif. Intell. Rev.*, 14(6):533–567. pages

Li, W., Han, J., and Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM*, pages 369–376. IEEE Computer Society. pages

Liu, B., Hsu, W., and Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 80–86. AAAI Press. pages

Mannila, H. (1997). Inductive databases and condensed representations for data mining. In *ILPS*, pages 21–30. pages

Mannila, H. (2000). Theoretical frameworks for data mining. *SIGKDD Explor. Newsl.*, 1:30–32. pages

Mannila, H. and Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov.*, 1(3):241–258. pages

Matys, V., Fricke, E., Geffers, R., GöSSling, E., Haubrock, M., Hehl, R., Hornischer, K., Karas, D., Kel, A. E., Kel-margoulis, O. V., Kloos, D.-U., Land, S., Lewicki-potapov, B., Michael, H., Münch, R., Reuter, I., Rotert, S., Saxel, H., Scheer, M., Thiele, S., and Wingender, E. (2003). TRANSFAC: transcriptional regulation, from patterns to profiles. *Nucleic Acids Research*, 31:374–378. pages

Meo, R., Psaila, G., and Ceri, S. (1998). An extension to sql for mining association rules. *Data Min. Knowl. Discov.*, 2(2):195–224. pages

Métivier, J.-P., Boizumault, P., Crémilleux, B., Khiari, M., and Loudni, S. (2011). A constraint-based language for declarative pattern discovery. In *Proceedings of the International ICDM Workshop on Declarative Pattern Mining*. IEEE Computer Society. To appear. pages

Mielikäinen, T. and Mannila, H. (2003). The pattern ordering problem. In *Knowledge Discovery in Databases, European Conference (PKDD 2003)*, volume 2838 of *Lecture Notes in Computer Science*, pages 327–338. Springer. pages

Morimoto, Y., Fukuda, T., Matsuzawa, H., Tokuyama, T., and Yoda, K. (1998). Algorithms for mining association rules for binary segmentations of huge categorical databases. In *Proceedings of 24rd International Conference on Very Large Data Bases*, pages 380–391. Morgan Kaufmann. pages

Morishita, S. and Sese, J. (2000). Traversing itemset lattice with statistical metric pruning. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 226–236. ACM. pages

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer. pages

Ng, R. T., Lakshmanan, L. V. S., Han, J., and Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 13–24, New York, NY, USA. ACM. pages

Nijssen, S. and Guns, T. (2010). Integrating constraint programming and itemset mining. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD-10)*, volume 6322 of *Lecture Notes in Computer Science*, pages 467–482. Springer. pages

Nijssen, S., Guns, T., and De Raedt, L. (2009). Correlated itemset mining in ROC space: A constraint programming approach. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-09)*, pages 647–656. ACM. pages

Nijssen, S. and Kok, J. N. (2005). Multi-class correlated pattern mining. In *Knowledge Discovery in Inductive Databases*, volume 3933 of *Lecture Notes in Computer Science*, pages 165–187. Springer. pages

Ohrimenko, O., Stuckey, P. J., and Codish, M. (2007). Propagation = lazy clause generation. In *Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer. pages

Parida, L. and Ramakrishnan, N. (2005). Redescription mining: Structure theory and algorithms. In *Proceedings, The Twentieth National Conference on Artificial Intelligence*, pages 837–844. AAAI Press. pages

Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer. pages

Pei, J. and Han, J. (2000). Can we push more constraints into frequent pattern mining? In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 350–354. ACM. pages

Pei, J., Han, J., and Lakshmanan, L. V. S. (2001). Mining frequent item sets with convertible constraints. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 433–442. IEEE. pages

Pei, J., Han, J., and Mao, R. (2000). Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30. ACM. pages

Perron, L. (1999). Search procedures and parallelism in constraint programming. In *Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*, pages 346–360. Springer. pages

Puget, J.-F. (1992). Pecos a high level constraint programming language. In *Singapore International Conference on Intelligent Systems (SPICIS)*. pages

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266. pages

Reischuk, R. M., Schulte, C., Stuckey, P. J., and Tack, G. (2009). Maintaining state in propagation solvers. In *Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 692–706, Berlin, Heidelberg. Springer. pages

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, New York, NY, USA. pages

Ruggieri, S. (2010). Frequent regular itemset mining. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–272. ACM. pages

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., and Tsuda, K. (2009). gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75:69–89. pages

Sandelin, A., Alkema, W., Engström, P. G., Wasserman, W. W., and Lenhard, B. (2004). Jaspar: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, 32:91–94. pages

Sandve, G., Abul, O., and Drabløs, F. (2008). Compo: composite motif discovery using discrete models. *BMC bioinformatics*, 9(1). pages

Schulte, C. (1999). Comparing trailing and copying for constraint programming. In *In Proceedings of the International Conference on Logic Programming*, pages 275–289. The MIT Press. pages

Schulte, C. (2002). *Programming Constraint Services: High-Level Programming of Standard and New Constraint Services*, volume 2302 of *Lecture Notes in Computer Science*. Springer. pages

Schulte, C. and Carlsson, M. (2006). Finite domain constraint programming systems. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 14, pages 495–526. Elsevier Science Publishers, Amsterdam, The Netherlands. pages

Schulte, C. and Stuckey, P. J. (2008). Efficient constraint propagation engines. *ACM Trans. Program. Lang. Syst.*, 31:2:1–2:43. pages

Sese, J. and Morishita, S. (2002). Answering the most correlated n association rules efficiently. In *Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Computer Science*, pages 410–422. Springer. pages

Sharan, R., Ovcharenko, I., Ben-Hur, A., and Karp, R. M. (2003). Creme: a framework for identifying cis-regulatory modules in human-mouse conserved segments. *Bioinformatics*, 19 (Suppl 1). pages

Shaw, P., De Backer, B., and Furnon, V. (2002). Improved local search for cp toolkits. *Annals of Operations Research*, 115:31–50. pages

Siebes, A., Vreeken, J., and van Leeuwen, M. (2006). Item sets that compress. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 393–404. SIAM. pages

Soulet, A. and Crémilleux, B. (2005). An efficient framework for mining flexible constraints. In *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *Lecture Notes in Computer Science*, pages 43–64. Springer. pages

Stuckey, P. J. (2011). Minizinc challenge 2011 results. In *Proceedings of the 1st international MiniZinc workshop (MZN'11)*. pages

Sun, H., De Bie, T., Storms, V., Fu, Q., Dhollander, T., Lemmens, K., Verstuyf, A., De Moor, B., and Marchal, K. (2009). ModuleDigger: an itemset mining framework for the detection of *cis*-regulatory modules. *BMC Bioinformatics*, 10 (Suppl 1). pages

Tan, P.-N. and Kumar, V. (2001). Mining indirect associations in web data. In *Mining Web Log Data Across All Customers Touch Points, Third International Workshop, Revised Papers*, volume 2356 of *Lecture Notes in Computer Science*, pages 145–166. Springer. pages

Tang, Z. and MacLennan, J. (2005). *Data mining with SQL Server 2005*. Wiley. pages

Thoma, M., Cheng, H., Gretton, A., Han, J., Kriegel, H.-P., Smola, A. J., Song, L., Yu, P. S., Yan, X., and Borgwardt, K. M. (2009). Near-optimal supervised feature selection among frequent subgraphs. In *Proceedings of the SIAM International Conference on Data Mining*, pages 1075–1086. SIAM. pages

Tompa, M., Li, N., Bailey, T. L., Church, G. M., De Moor, B., Eskin, E., Favorov, A. V., Frith, M. C., Fu, Y., Kent, W. J., Makeev, V. J., Mironov, A. A., Noble, W. S., Pavesi, G., Pesole, G., Régnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., and Zhu, Z. (2005). Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23:137–144. pages

Turi, A., Loglisci, C., Salvemini, E., Grillo, G., Malerba, D., and D'Elia, D. (2009). Computational annotation of UTR cis-regulatory modules through frequent pattern mining. *BMC Bioinformatics*, 10 (Suppl 6). pages

Uno, T., Kiyomi, M., and Arimura, H. (2005). Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining*, pages 77–86. ACM. pages

Van Hentenryck, P. (1999). *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA. pages

Van Hentenryck, P. and Deville, Y. (1993). *The cardinality operator: A new logical connective for constraint logic programming*, pages 383–403. MIT Press, Cambridge, MA, USA. pages

Van Hentenryck, P., Deville, Y., and Teng, C.-M. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2-3):291–321. pages

Van Hentenryck, P. and Michel, L. (2009). *Constraint-Based Local Search*. MIT Press. pages

Van Hentenryck, P., Perron, L., and Puget, J.-F. (2000). Search and strategies in OPL. *ACM Trans. Comput. Log.*, 1(2):285–320. pages

Van Hentenryck, P., Saraswat, V. A., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(FD). *J. Log. Program.*, 37(1-3):139–164. pages

Van Loo, P., Aerts, S., Thienpont, B., De Moor, B., Moreau, Y., and Marynen, P. (2008). Moduleminer - improved computational detection of cis-regulatory modules: are there different modes of gene regulation in embryonic development and adult tissues? *Genome Biology*, 9(4):R66+. pages

Webb, G. (2007). Discovering significant patterns. *Machine Learning*, 68:1–33. 10.1007/s10994-007-5006-x. pages

Winston, W. and Venkataramanan, M. (2003). *Introduction to Mathematical Programming: Applications and Algorithms*. Cengage Learning. pages

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In *Principles of Data Mining and Knowledge Discovery*, volume 1263 of *Lecture Notes in Computer Science*, pages 78–87. Springer. pages

Xie, D., Cai, J., Chia, N.-Y., Ng, H., and Zhong, S. (2008). Cross-species de novo identification of cis-regulatory modules with GibbsModule: application to gene regulation in embryonic stem cells. *Genome Research*, 18(8):1325–35. pages

Yan, X., Cheng, H., Han, J., and Xin, D. (2005). Summarizing itemset patterns: a profile-based approach. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 314–323. ACM. pages

Yang, Q. and Wu, X. (2006). 10 Challenging Problems in Data Mining Research. *International Journal of Information Technology & Decision Making*, 5(4):597–604. pages

Zaki, M. J. and Gouda, K. (2003). Fast vertical mining using diffsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 326–335. ACM. pages

Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. (1997). New algorithms for fast discovery of association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 283–286. AAAI Press. pages

Zhao, L., Zaki, M. J., and Ramakrishnan, N. (2006). BLOSOM: A framework for mining arbitrary boolean expressions. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, pages 827–832. ACM. pages

List of publications

Journal articles

- T. Guns, S. Nijssen, and L. De Raedt. *k-pattern set mining under constraints*, IEEE Transactions on Knowledge and Data Engineering, To appear, 2011.
  Winner of the Innovation Award at the Lorentz workshop on Mining Patterns and Subgroups, 2010.
- T. Guns, S. Nijssen, and L. De Raedt. *Itemset mining: A constraint programming perspective*, Artificial Intelligence, volume 175(12-13):1951–1983, 2011.
  http://dx.doi.org/10.1016/j.artint.2011.05.002

Book chapters

- J. Besson, J.F. Boulicaut, T. Guns, S. Nijssen. *Generalizing itemset mining in a constraint programming setting*, in S. Dzeroski, B. Goethals, P. Panov, editors, *Inductive Databases and Constraint-Based Data Mining*, pages 107-126, Springer, 2010.
  http://dx.doi.org/10.1007/978-1-4419-7738-0_5

Conference papers

- T. Guns, S. Nijssen, and L. De Raedt. *Evaluating pattern set mining strategies in a constraint programming framework*, Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-11), Shenzhen, China, 24-27 May 2011, in Advances in Knowledge Discovery

and Data Mining, volume 6635 of Lecture Notes in Computer Science, pages 382–194, Springer, 2011.
http://dx.doi.org/10.1007/978-3-642-20847-8_32

- T. Guns, H. Sun, K. Marchal, and S. Nijssen. *Cis-regulatory module detection using constraint programming*, IEEE International Conference on Bioinformatics & Biomedicine (BIBM-10), Hong Kong, 18-21 December 2010, in Proceedings of the IEEE International Conference on Bioinformatics & Biomedicine, pages 363–368, IEEE Computer Society, 2010.
http://dx.doi.org/0.1109/BIBM.2010.5706592

- S. Nijssen and T. Guns. *Integrating constraint programming and itemset mining*, European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD-10), Barcelona, 20-24 September 2010, in Machine Learning and Knowledge Discovery in Databases, volume 6322 of Lecture Notes in Computer Science, pages 467–482, Springer, 2010.
http://dx.doi.org/10.1007/978-3-642-15883-4_30

- L. De Raedt, T. Guns, and S. Nijssen. *Constraint programming for data mining and machine learning*, AAAI Conference on Artificial Intelligence (AAAI-10), Atlanta, Georgia, USA, 11-15 July 2010, in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pages 1671–1675, AAAI Press, 2010.
http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1837

- S. Nijssen, T. Guns, and L. De Raedt. *Correlated itemset mining in ROC space: A constraint programming approach*, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-09), Paris, France, 28 June - 1 July 2009, in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 647–656, ACM Press, 2009.
http://dx.doi.org/10.1145/1557019.1557092

- L. De Raedt, T. Guns, and S. Nijssen. *Constraint programming for itemset mining*, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-08), Las Vegas, Nevada, USA, 24-27 August 2008, in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 204–212, ACM, 2008.
http://dx.doi.org/10.1145/1401890.1401919

Workshop papers and posters

- S. Nijssen, A. Jiménez, T. Guns. *Constraint-based pattern mining in multi-relational databases*, in Proceedings of the International ICDM Workshop on Declarative Pattern Mining, IEEE Computer Society, to appear, 2011.
- T. Guns, S. Nijssen, A. Zimmermann, L. De Raedt. *Declarative heuristic search for pattern set mining*, in Proceedings of the International ICDM Workshop on Declarative Pattern Mining, IEEE Computer Society, to appear, 2011.
- T. Guns. *Demo of the gecode-based constraint programming for itemset mining system*, Lorentz workshop on Mining Patterns and Subgroups, Lorentz center, Leiden, 2010 (demo).
- T. Guns, S. Nijssen, L. De Raedt. *k-Pattern set mining under constraints*, Lorentz workshop on Mining Patterns and Subgroups, Lorentz center, Leiden, 2010.
  Winner of the Innovation Award.
- T. Guns, H. Sun, S. Nijssen, A.S. Rodriguez, L. De Raedt, K. Marchal. *Proximity-based cis-regulatory module detection using constraint programming for itemset mining*, European Conference on Computational Biology, Ghent, Belgium, 2010 (poster).
- T. Guns, S. Nijssen, L. De Raedt. *Constraint programming for correlated itemset mining*, Benelux Conference on Artificial Intelligence, Eindhoven, the Netherlands, 29-30 October 2009, Proceedings of the 21st Benelux Conference on Artificial Intelligence, pages 315-316, 2009.
  http://wwwis.win.tue.nl/bnaic2009/proc.html
- T. Guns. *Constraint programming for itemset mining*, Summer School on the Analysis of Patterns, Cagliari, Italy, 2009 (seminar).

Technical reports

- T. Guns, S. Nijssen, and L. De Raedt. *k-pattern set mining under constraints*, CW Reports, volume CW596, pages 1–26, Department of Computer Science, K.U.Leuven, Belgium, 2010.

Curriculum Vitae

Tias Guns studied computer science (licentiaat informatica) at the University of Leuven (KU Leuven), Belgium. His master thesis was titled 'A genetic algorithm for the construction and training of layered neural networks' and was supervised by professor Hendrik Blockeel. He graduated with great honours in June 2006.

In August 2006 he endeavoured on a year-long pedagogical training in circus arts at the Brussels circus school (Ecole de Cirque de Bruxelles), and has been teaching circus skills to youngster ever since.

In september 2007, he joined the Declarative Languages and Artificial Intelligence group at the University of Leuven to start his doctural studies. In Januari 2008 he received a four year personal grant from the Institute for the promotion of innovation through science and technology in Flanders (IWT Vlaanderen) to work on his Ph.D. titled "Declarative Pattern Mining using Constraint Programming".

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Declarative Languages and Artificial Intelligence
Arenbergkasteel
B-3000 Leuven