Fouille de données déclarative basée sur la programmation par contraintes.

Mohamed-Bachir Belaid

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche : LIRMM

Declarative Itemset Mining Based on Constraint Programming

Présentée par Mohamed-Bachir BELAID

Le 08/01/2020

Sous la direction de Christian BESSIERE et Nadjib LAZAAR

Devant le jury composé de

Samir LOUDNI, MdC, HDR, Université de Caen Normandie	Rapporteur
Jilles VREEKEN, Senior Researcher, Max Planck Institute for Informatics and Saarland University	Rapporteur
Thi-Bich-Hanh DAO, MdC, HDR, Université d'Orléans	Examinatrice
Christine SOLNON, Professeure, INSA de Lyon	Présidente
Christian BESSIERE, DR, CNRS, Université de Montpellier	Directeur de thèse
Nadjib LAZAAR, MdC, Université de Montpellier	Encadrant

UNIVERSITÉ DE MONTPELLIER

To Ommi, my mother....

Abstract

Data mining is the art of discovering knowledge from databases. The user specifies the type of patterns to be mined, and the miner uses techniques to find the required patterns. Many techniques have been introduced for mining traditional patterns like frequent itemsets, association rules, etc. However, mining patterns with additional properties remains a bottleneck for specialists nowadays due to the algorithmic effort needed to handle these properties.

Recently, researchers have taken advantage of the flexibility of constraint programming to model various data mining problems. In terms of CPU time, constraint programming-based methods have not yet competed with ad hoc algorithms. However, their flexibility allows the modeling of complex user queries without revising the solving process.

In this thesis we propose to use constraint programming for modeling and solving some well known data mining problems. Our first contribution is a constraint programming model for mining association rules. To implement our model, we introduce a new global constraint, CONFIDENT, for ensuring the confidence of rules. We prove that completely propagating CONFIDENT is NP-hard. We thus provide a non-complete propagator and a decomposition for CONFIDENT. We also capture the minimal non-redundant rules, a condensed representation of association rules, by introducing the global constraint GENERATOR. GENERATOR is used for mining itemsets that are generators. For this constraint, we propose a complete polynomial propagator.

Our second contribution is a generic framework based on constraint programming to mine both borders of frequent itemsets, i.e. the positive border or *maximal frequent itemsets* and the negative border or *minimal infrequent itemsets*. One can easily decide which border to mine by setting a simple parameter. For this, we introduce two new global constraints, FREQUENTSUBS and INFREQUENTSUPERS, with complete polynomial propagators. We then consider the problem of mining borders with additional constraints. We prove that this problem is coNP-hard, ruling out the hope for the existence of a single CSP solving this problem (unless coNP $\subseteq$ NP).

Résumé

La fouille de données est l'art de découvrir des informations à partir de bases de données. L'utilisateur spécifie le type de motifs à extraire et le spécialiste utilise des techniques pour trouver les motifs requis. De nombreuses techniques ont été introduites pour l'extraction des motifs clasiques tels que les motifs fréquents, les règles d'association, etc. Cependant, l'extraction des motifs avec des propriétés supplémentaires restent un problème pour les spécialistes car des efforts algorithmiques sont requises pour gérer ces propriétés.

Récemment, les chercheurs ont profité de la flexibilité de la programmation par contraintes pour modéliser plusieurs problèmes de la fouille de données. En termes de temps d'exécution, les méthodes basées sur la programmation par contraintes ne sont pas encore concurrentes avec les algorithmes specialisées. Cependant, leur flexibilité permet la modélisation des requêtes complexes sans la nécessité de réviser le processus de résolution.

Dans cette thèse, nous proposons d'utiliser la programmation par contraintes pour résoudre des problèmes de la fouille de données. Notre première contribution est un modèle basé sur la programmation par contraintes pour l'extraction des règles d'association. Pour mettre en œuvre notre modèle, nous introduisons une nouvelle contrainte globale, CONFIDENT, pour assurer la confiance des règles. Nous prouvons que propager complètement CONFIDENT est NP-difficile. Nous fournissons donc un propagateur non-complet et une décomposition pour la contrainte CONFIDENT. Nous capturons également les règles minimales non redondantes, une représentation condensée des règles d'association, en introduisant la contrainte globale GENERATOR. GENERATOR est utilisé pour extraire des motifs qui sont des générateurs. Pour cette contrainte, nous proposons un propagateur polynomial complet.

Notre deuxième contribution est un model générique basé sur la programmation par contraintes permettant l'extraction des deux frontières des motifs fréquents, à savoir la frontière positive ou *les motifs maximaux fréquents* et la frontière négative ou *les motifs minimaux infréquents*. Il est facile de choisir la frontière à extraire en fixant un simple paramètre. Pour cela, nous introduisons deux nouvelles contraintes globales, FREQUENTSUBS et INFREQUENTSUPERS, avec des propagateurs polynomiaux complets. Nous examinons ensuite le problème de l'extraction des frontières avec des contraintes supplémentaires.

Nous prouvons que ce problème est coNP-difficile. Cela implique qu'il n'existe aucun CSP représentant ce problème (sauf si coNP $\subseteq$ NP).

Acknowledgements

This work could not be possible without the help of some persons in my entourage.

First, special thanks to my family members for their moral support, my sister Hadjer, my brother Said and my mother to whom I dedicate this thesis.

I thank my supervisors Christian BESSIERE and Nadjib LAZAAR for all their directions and contributions in this work. Thanks to members of my team COCONUT and to all members of my lab LIRMM.

I also had some interesting discussions with scientists from different labs. I thank Said JABBOUR and Lakhdare SAIS for hosting me in their lab, CRIL. Thanks to Yahia LABBAH, Mehdi MAAMAR Tias GUNS and Pierre SCHAUS for all the discussions that we had.

Finally I thank the jury members, Samir LOUDNI, Jilles VREEKEN, Thi-Bich-Hanh DAO and Christine SOLNON for evaluating this work.

Introduction

Data mining is the process of transforming a raw data into useful information. Itemset mining is one of the most studied problems in data mining. It was originally introduced for sales transactions and products (Agrawal et al. [1993]). In a transaction dataset one may look for significant regularities like *"the product A is bought frequently"* or *"people buying A usually buy B with it"*.

Motivations

Many techniques have been developed for itemset mining problems including the mining of frequent itemsets (Agrawal et al. [1993], Han et al. [2000]), mining association rules (Agrawal et al. [1993], Szathmary et al. [2007a]), mining closed itemsets (Pasquier et al. [1999]), mining high utility itemsets (Chan et al. [2003]) etc. These tasks become easier and easier thanks to dedicated algorithms and thanks to the improvement of computers performance. However, the number of extracted patterns [1] is often huge and can easily exceed the size of the dataset itself. In this case the user faces an enormous number of irrelevant patterns. However, most of the time the user is interested in patterns that satisfy some specific properties. For instance, the user may ask for patterns containing/not containing some specific items.

According to Wojciechowski and Zakrzewicz [2002], there are three ways to handle the additional user's constraints. We can use a pre-processing step that restricts the dataset to only transactions that satisfy the constraints. Such a technique cannot be used on all kinds of constraints. We can use a post-processing step to filter out the patterns violating the user's constraints. Such a brute-force technique can be computationally infeasible when the problem without the user's constraints has too many solutions. We can finally integrate the filtering of the user's constraints into the specialized data mining process in order to extract only the patterns satisfying the constraints. Such a technique requires the development of a new algorithm for each new problem with user's constraints.

1. In this manuscript we restrict the use of *p*attern to itemsets and/or association rules.

In a recent line of work, declarative approaches are used to solve various itemset mining problems (Raedt et al. [2008], Khiari et al. [2010], Boudane et al. [2016], Lazaar et al. [2016], Schaus et al. [2017], Bessiere et al. [2018]). In terms of CPU time, declarative methods have not yet competed with ad hoc algorithms. However, their flexibility allows the modeling of complex user queries without revising the solving process. One of the most powerful declarative approaches is *constraint programming*. In Constraint Programming (CP) the user specifies his problem and the computer solvers it.

Contributions

In this thesis, we propose to use constraint programming techniques to solve some itemset mining problems. We propose generic models, new global constraints with propagators and we study their effectiveness. We also study problems of mining patterns in the presence of additional constraints. The contributions of this thesis is fourfold.

1. **Modeling:** In this thesis, we propose constraint programming based models to solve some itemset mining problems. We propose a constraint programming model for mining association rules that are frequent and confident. Then, we show that this model can be extended to mine only the Minimal Non-redundant Rules (MNRs), a condensed representation for association rules. We propose a generic constraint programming for mining borders of frequent itemsets, i.e. the positive border or *maximal frequent itemsets* and the negative border or *minimal infrequent itemsets*. One can easily decide which border to mine using a simple parameter. We show that our model can be extended to consider additional user's constraints.

2. **Solving:** In this thesis, we do not only model problems using existing constraints but we also contribute at the solving part of constraint programming by introducing new global constraints with their filtering algorithms (aka, propagators). We introduce a new global constraint CONFIDENT. CONFIDENT is used to ensure the confidence of rules. We prove that completely propagating CONFIDENT is NP-hard. Thus, we provide CONFIDENT with two different propagators. The first one uses a non-complete filtering rule. The second one uses a decomposition using existing constraints. We introduce a new global constraint GENERATOR for mining generator itemsets. This constraint can be used to mine MNRs. We provide GENERATOR with a polynomial filtering algorithm. We prove that this algorithm is complete.

   We introduce two new global constraints, (1) FREQUENTSUBS for mining itemsets having only frequent subsets and (2) INFREQUENTSUPERS for mining itemsets having only infrequent supersets. We provide both constraints with complete polynomial propagators. These constraints are used in our generic model for mining borders of frequent itemsets.

   We show that existing propagators for the infrequency are not complete. Hence, we propose a new global constraint INFREQUENT for mining infrequent itemsets with a complete polynomial algorithm and we evaluate its efficiency.

3. **Studying constrained patterns:** In this thesis, we address the issue of mining patterns in the presence of other constraints. As noticed in (Bonchi and Lucchese

[2004]), mining borders under additional constraints has two interpretations, (1) borders that, in addition, satisfy the set of constraints (2) borders of the itemsets satisfying the constraints. The user is usually interested in mining itemsets w.r.t. the second interpretation. This is true on positive/negative borders, but also on closed or generator itemsets. We prove that it is coNP-hard to find maximal/minimal itemsets among those satisfying a set of additional constraints, i.e. solve the problem w.r.t. the second interpretation. This implies that there does not exist any CSP representing the problem of finding borders under additional constraints unless coNP $\subseteq$ NP.

This problem can occur also on mining MNRs with additional constraints. We prove that in a specific case both interpretations (1 and 2) are equivalent.

4. **Implementing:** To evaluate their efficiency, we have implemented our models using the constraint solver Oscar. For this, we have used the already implemented constraints and we have implemented the propagators of our new global constraints. Results can be found in the experiment sections in the contributions part.

Manuscript organisation

This manuscript is organised in two parts, Part I for the background and Part II for the contributions. Every part is organised in chapters:

I **Background:**
- **Chapter 1:** In this chapter we present the basic methods and notations in itemset mining. We introduce the different tasks in itemset mining, frequent itemsets mining, condensed representations mining, association rules mining, etc. We also give an overview on other tasks of data mining.
- **Chapter 2:** In this chapter we present the constraint programming paradigm. We present some basic definitions and notations. We also present the two main features of constraint solvers, i.e. the backtracking and the propagation processes.
- **Chapter 3:** In this chapter we present some existing constraint-based methods to solve itemset mining problems. We focus on the constraints based approaches, i.e. the basic constraint programming model, global constraints and the SAT approaches for itemset mining.

II **Contributions:**
- **Chapter 4:** In this chapter we introduce our first contribution, a constraint programming model for mining association rules with a new global constraint CONFIDENT. We show that this model can be easily extended to mine minimal non-redundant rules using a new global constraint GENERATOR. We also study the problem of combining minimal non-redundant rules with constraints.
- **Chapter 5:** Finally, we introduce our second contribution, a generic constraint programming model for mining borders of frequent itemsets, i.e. the positive

border or *maximal frequent itemsets* and the negative border or *minimal infrequent itemsets*. For this we introduce two global constraints with complete polynomial propagators, FREQUENTSUBS and INFREQUENTSUPERS. We also study the problem of combining condensed representations with constraints.

List of publications

— Belaid et al. [2018]: In proceedings of the doctoral program at the international conference on principles and practice of constraint programming 2018 in Lille, France.
— Belaid et al. [2019b]: In proceedings of the SIAM international conference on data mining 2019 in Calgary, Canada.
— Belaid et al. [2019a]: In proceedings of the international joint conference on artificial intelligence 2019 in Macau, China.

Contents

List of Figures

List of Tables

Part I

Background

I

Chapter 1: Itemset Mining

1.1 Introduction

Itemset mining is one of the most studied problems in data mining. It was originally introduced for sales transactions and products (Agrawal et al. [1993]).

In this chapter, we introduce the basic concepts of itemset mining. We start with some basic definitions in Section 1.2. In Section 1.3 we review some methods for mining frequent and infrequent itemsets. Condensed representations are presented in Section 1.4. Association rules mining, its methods and variants are presented in Section 1.5. We define the problem of mining constrained itemsets in Section 1.6. A data compression technique is presented in Section 1.7. Finally, sequence and uncertain data mining are presented in Section 1.8.

1.2 Notations and Definitions

1.2.1 Itemset Mining

Let $\mathcal{I} = \{1, \ldots, n\}$ be a set of n *item* indices and $\mathcal{T} = \{1, \ldots, m\}$ a set of m *transaction* indices. An itemset P is a non empty subset of $\mathcal{I}$. The transactional dataset $\mathcal{D} = \{\{t_1, \ldots, t_m\}\}$ is a multiset of itemsets, where for all $i \in \mathcal{T}$, t_i is an itemset. The cover of an itemset P, denoted by $cover(P)$, is the set of transaction indices i for which t_i contains P. Given S a subset of $\mathcal{T}$, $item(S)$ is the set of items belonging to all transactions whose index is in S. The frequency of an itemset P is the cardinality of its $cover$, i.e. $freq(P) = |cover(P)|$. Let s be some given constant called a *frequency threshold*. The itemset P is *frequent* if $freq(P) \geq s$. P is *infrequent* (or *rare*) if $freq(P) < s$. The closure of an

itemset P is the set of items that belong to all transactions whose index is in $cover(P)$, i.e. $closure(P) = item(cover(P))$. P is closed iff $closure(P) = P$. We denote by P^f the itemset P with frequency f.

Table 1.1 – *Transaction dataset example with five items and five transactions. The standard representation (left) and the binary representaiton (right).*

trans.	Items
t_1	$A\ B\ \ D\ E$
t_2	$A\ \ C$
t_3	$A\ B\ C\ \ E$
t_4	$\ B\ C\ \ E$
t_5	$A\ B\ C\ \ E$

	A	B	C	D	E
t_1	1	1	0	1	1
t_2	1	0	1	0	0
t_3	1	1	1	0	1
t_4	0	1	1	0	1
t_5	1	1	1	0	1

Example 1.1. The dataset in Table 1.1 has 5 items, i.e. $|\mathcal{I}|$ = 5 and 5 transactions, i.e. $|\mathcal{T}| = 5$. The cover of CE is $cover(CE) = \{3, 4, 5\}$. Its frequency is the cardinality of its cover, i.e. $freq(CE) = |cover(CE)| = 3$. We denote it CE^3. With $s = 3$, the itemset BC is frequent ($freq(BC) = 3 \geq 3$). The itemset AD is infrequent ($freq(AD) = 1 < 3$). The closure of AB is $closure(AB) = ABE$ that is because $cover(AB) = \{1, 3, 5\}$ and ABE belongs to all transactions whose index is in $cover(AB)$.

1.2.2 Association Rules

An *association rule* is an implication of the form $X \rightarrow Y$, where X and Y are itemsets such that $X \cap Y = \emptyset$ and $Y \neq \emptyset$. X represents the *body* of the rule and Y represents its *head*. The frequency of a rule $X \rightarrow Y$ is the frequency of the itemset $X \cup Y$, that is, $freq(X \rightarrow Y) = freq(X \cup Y)$. The *confidence* of a rule captures how often Y occurs in transactions containing X, that is, $conf(X \rightarrow Y) = \frac{freq(X \rightarrow Y)}{freq(X)}$. Given a minimum confidence c, a rule $X \rightarrow Y$ is confident if $conf(X \rightarrow Y) \geq c$. A rule $X \rightarrow Y$ is valid if it is frequent w.r.t. s and confident w.r.t. c.

Example 1.2. Consider the transaction dataset presented in Table 1.1. With $s = 3$ and $c = 60\%$, $B \rightarrow C$ is a valid association rule because $freq(B \rightarrow C) = 3 \geq s$ and $conf(B \rightarrow C) = \frac{freq(B \rightarrow C)}{freq(B)} = 75\% \geq c$.

1.3 Frequent Itemsets

Mining frequent itemsets is the problem of listing the set of all itemsets with a frequency above a given threshold. Mining frequent itemsets is essential for many data mining tasks. It captures recurrent phenomena in a dataset. In this section we review some well known algorithms for mining frequent itemsets. We then define the problems of mining frequent itemsets with variable threshold. We study the negation of the problem of mining frequent itemsets, i.e. mining infrequent itemsets. Finally, we review the problem of mining frequent itemsets from incremental datasets.

1.3.1 Methods for Mining Frequent Itemsets

Apriori principal

The APRIORI algorithm was first introduced in (Agrawal et al. [1993]) for mining association rules. Since then many APRIORI based algorithms were proposed (Park et al. [1995], Ye and Chiang [2006], Lin et al. [2012], Li et al. [2012]). The basic idea of APRIORI is to generate frequent itemsets of length $k+1$ from those of length k. Before the generation APRIORI applies some pruning using the anti-monotony property of the frequency.

Proposition 1.1 (Anti-monotony property of the frequency). If the itemset X is infrequent, then any itemset Y such that $Y \supsetneq X$, is infrequent too.

Proposition 1.1 implies that if X is frequent then any itemset Y such that $Y \subsetneq X$, is frequent too.

APRIORI starts with items ($k = 1$) and generates itemsets of length 2 and so on until no itemset is to be mined. Thanks to Proposition 1.1, infrequent items will not participate at generating the itemsets of length 2.

Let L_k be the list of frequent itemsets of size k and C_k the list of candidate itemsets of size k. At iteration k, APRIORI generates C_k from L_{k-1} then prunes infrequent itemsets to create the list of frequent itemsets of size k, i.e. L_k. This process is repeated until L_k is empty. Finally, The set of frequent itemsets corresponds to the union of all lists, i.e. $\cup_{k=0}^{n} L_k$.

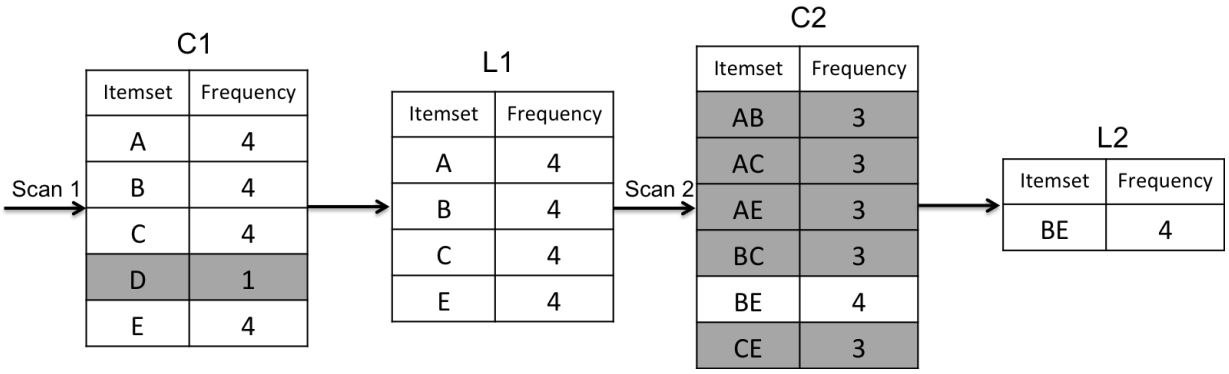


Figure 1.1 – APRIORI *on the dataset in Table 1.1 with* $s = 4$.

Example 1.3. In Figure 1.1 the first scan generates the candidates of length 1, i.e. C_1. The item D is then pruned due its infrequency. As a result, D is not considered in further generations thanks to Proposition 1.1 (no superset of D can be frequent). This results the list of frequent 1-itemsets (L_1) from which itemsets of length 2 are generated. The same process is repeated for $k = 2$.

Despite the extensive pruning conducted by APRIORI, it has some limits. First, it is costly to handle a huge number of candidate sets specially in terms of memory. If the dataset has 10^4 items, one should generate more than 10^7 candidate itemsets for $k = 2$. Second, APRIORI requires a scan of the dataset for every iteration, which can be tedious.

FP-GROWTH algorithm

To get over APRIORI's limits, the FP-GROWTH algorithm was proposed in (Han et al. [2000]). FP-GROWTH claims that it is an algorithm for mining frequent itemsets without candidates generation. FP-GROWTH uses a compact data structure called FP-tree (Frequent Pattern tree) to mine frequent itemsets.
The first step of FP-GROWTH is to create the FP-tree data structure. Then one can use the FP-tree to generate frequent itemsets. We illustrate both steps in the following example.

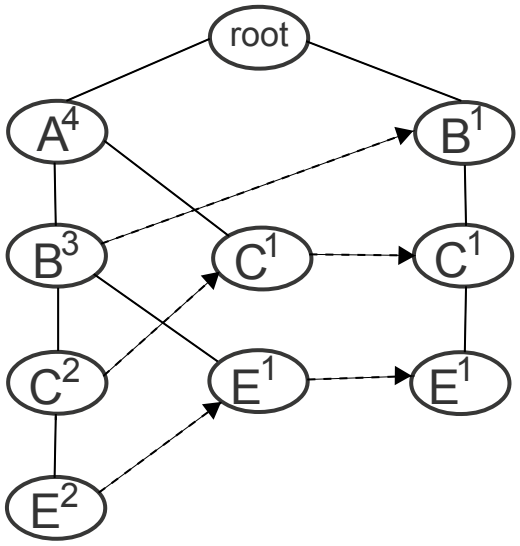


Figure 1.2 – *The FP-tree of the dataset in Table 1.1 with* $s = 3$.

Example 1.4. In Figure 1.2 we present the FP-tree of the dataset in Table 1.1 with $s = 3$. The item D is infrequent and hence not considered in the FP-tree creation. Usually the order of the items is considered according to their frequencies. As all frequent items have the same frequency, we consider the lexicographic order $\langle A, B, C, E\rangle$.

Step 1: FP-tree creation: We start by mapping the transaction t_1 to the tree by constructing the path $root \rightarrow A^1 \rightarrow B^1 \rightarrow E^1$. The second transaction has the same prefix as the first one (A). In this case, the frequency count is incremented for A and a node is created for C ($root \rightarrow A^2 \rightarrow C^1$). The transaction t_3 has the prefix AB that was already created thanks to transaction t_1. We simply increment the frequency counts for A and B and create the nodes C and E with a frequency count of 1 ($root \rightarrow A^3 \rightarrow B^2 \rightarrow C^1 \rightarrow E^1$). The transaction t_4 has no common prefix with the already created paths, in this case we create the new path $root \rightarrow B^1 \rightarrow C^1 \rightarrow E^1$. The transaction t_5 overlaps with an already existing path. We simply increment the frequency count for every node in this path ($root \rightarrow A^4 \rightarrow B^3 \rightarrow C^2 \rightarrow E^2$). Finally, nodes with the same item are connected with dashed arrows.

Step 2: Mining frequent itemsets using the FP-tree: Using the FP-tree, one can

generate frequent itemsets by exploring the tree from the bottom. That is, we start with nodes in an inverted order. For every node, a set of frequent itemsets is extracted. The union of these sets is the set of all frequent itemsets.
From the node E one can generate the frequent itemsets ABE^3, BCE^3, BE^4, CE^3, AE^3 and E^4. The frequency of ABE, for instance, is identified by summing-up the frequency counts of the two paths containing AB in the prefix of E ($root \rightarrow A^4 \rightarrow B^3 \rightarrow C^2 \rightarrow E^2$ and $root \rightarrow A^4 \rightarrow B^3 \rightarrow E^1$), i.e. $freq(ABE) = 1 + 2 = 3$. The frequency of ACE is identified by the single path which contains AC in the prefix of E ($root \rightarrow A^4 \rightarrow B^3 \rightarrow C^2 \rightarrow E^2$), i.e. $freq(ACE) = 2$, i.e. ACE is infrequent.

1.3.2 Frequent Itemsets with Variable Threshold

Setting a single frequency threshold to a high value misses rare but important patterns. On the other hand, set it to a low value could generate many meaningless patterns. For instance, in a supermarket transaction data, buying a *car* is rare but very beneficial and patterns containing the item *car* can be of high importance. One can set the threshold to a low value to include this important event. However, this may include insignificant patterns like buying *milk* with *gum*.

To solve this problem, a number of strategies were proposed. One possible solution is to associate to every item a frequency threshold according to its nature. In this case an itemset is frequent if its frequency exceeds the minimum among thresholds of its items (Liu et al. [1999], Kiran and Reddy [2011]). Han and Fu [1995] associate to every level (itemsets with a given size) a frequency threshold and an itemset of size k is frequent if its frequency is no less than the threshold associated to the level k. In (Wang et al. [2000a]) frequency constraints are defined according to bins of items.

1.3.3 Infrequent Itemsets

Researchers have focused almost entirely on frequent itemset mining. However, infrequent itemsets can hold useful information in many real-world problems. Infrequent itemsets can give insights into rare phenomena. For instance, in the security field, normal behaviours are usually frequent, whereas suspicious behaviours are very rare. Infrequent itemsets can be used in several domains such as biology, medicine and security (Lee and Stolfo [1998], Masuda et al. [2002], Prati et al. [2003]). For example, in biology an expert may be interested in identifying the causes of the cardiovascular diseases (CVD). A rare pattern like "vegetarians, CVD" leads to the possible information "vegetarians are at a low risk of having CVD".

Most proposed methods for mining infrequent itemsets are based on APRIORI (Koh and Rountree [2005], Adda et al. [2007], Troiano et al. [2009]). In (Koh and Rountree [2005]) the authors present APRIORI-*Inverse* which prunes the search space by discarding all itemsets above the maximum frequency threshold. *AfRIM* (APRIORI for Rare Itemset Mining) (Adda et al. [2007]) uses the pruning principle of APRIORI by exploiting the monotonicity

property of the frequency. *Rarity* (Troiano et al. [2009]) starts by the longest possible infrequent itemset in a dataset and moves down through the lattice until no infrequent itemset is to be mined.

1.3.4 Mining Frequent Itemsets from Incremental Datasets

Mining frequent itemsets from an incremental dataset is the problem of mining frequent itemsets when we allow an update of the original dataset (Cheung et al. [1996], Ayad et al. [2001]). More formally, let $\mathcal{D}$ be the original dataset and L_s the list of frequent itemsets in $\mathcal{D}$ w.r.t. s, the relative frequency threshold (s is in percentage). We denote by $\mathcal{D}'$ the additional multiset of transactions. $\mathcal{D} \cup \mathcal{D}'$ is then the updated dataset. This problem consists in finding the set of frequent itemsets in $\mathcal{D} \cup \mathcal{D}'$, i.e. L'_s, from L_s.

In (Cheung et al. [1996]) an algorithm called FUP (Fast UPdate) was proposed for mining frequent itemsets from incremental datasets. FUP takes as input the frequency threshold s, the original dataset $\mathcal{D}$, $\mathcal{D}'$ and L_s and returns as output L'_s, the set of frequent itemsets in $\mathcal{D} \cup \mathcal{D}'$. FUP prunes losers (itemsets that are in L_s but become infrequent after the update) and adds winners (those that are not in L_s but become frequent after the update). The experiments showed the benefit of taking advantage of L_s for mining L'_s comparing to APRIORI which mines L'_s directly from $\mathcal{D} \cup \mathcal{D}'$.

1.4 Condensed representations

The number of frequent/infrequent itemsets can be huge, making it hard even to print the result. Hence, we often reduce the problem of mining frequent or infrequent itemsets to the problem of mining the *borders*. The *positive* border is the set of frequent itemsets with only infrequent supersets, i.e. Maximal Frequent Itemsets (MFIs). The *negative* border is the set of infrequent itemsets with only frequent subsets, i.e. Minimal Infrequent Itemsets (MIIs). The subsets of the MFIs and the supersets of the MIIs represent the frequent and infrequent itemsets, respectively.

Given the set of MFIs/MIIs one can reproduce all the frequent/infrequent itemsets. But it requires several scans of the data to identify their frequencies. The set of *closed/generator* itemsets allows the reproduction of frequent or infrequent itemsets with their frequencies. Closed (generator) itemsets are those maximal (minimal) w.r.t. a given frequency. In other words, a closed itemset is an itemset that has no superset with the same frequency and a generator is an itemset that has no subset with the same frequency.

Equivalence classes include itemsets that have the same frequency with an inclusion relation between them. More formally, the itemsets X and Y belong to the same *equivalence class* iff $closure(X) = closure(Y)$. The maximal itemset within an equivalence class is closed and the minimal is generator.

Table 1.2 – *Dataset with 5 items and 6 transactions (left) and associated prices for items (right)*

a

trans.	Items
t_1	$A\ B\ C\ D\ E$
t_2	$B\ C$
t_3	$B\ C\ D\ E$
t_4	$A\ B\ C\ D$
t_5	$A\ B\ C\ E$
t_6	$B\ C\ D\ E$

b

Item	Price
A	19
B	18
C	2
D	4
E	14

1.4.1 Maximal Frequent Itemsets

Maximal frequent itemsets represent the smallest set possible from which all frequent itemsets are reproduced. It is usually significantly smaller than the set of frequent itemsets. In (Mannila and Toivonen [1997]) MFIs are referred to as positive border.

Definition 1.1 (Maximal Frequent Itemset (MFI))**.** An itemset is an MFI if it is frequent and all its proper supersets are infrequent.

Example 1.5. With $s = 3$, the dataset in Table 1.2a has MFIs = $\{ABC^3, BCDE^3\}$. The dataset is illustrated in Figure 1.3. The itemset ABC is frequent ($freq(ABC) = 3 \geq 3$) and all its proper supersets are infrequent ($freq(ABCE) = 2 < 3$ and $freq(ABCD) = 2 < 3$).

Several techniques have been introduced for mining MFIs (Jr. [1998], Gouda and Zaki [2001], Burdick et al. [2001], Uno et al. [2004]). Most of these techniques adapt frequent itemsets mining methods for mining only MFIs (APRIORI, FP-GROWTH, ...).

Using MFIs, one can reproduce all frequent itemsets. But, unfortunately it requires several dataset scans to identify the frequency of every itemset. The set of MFIs is a subset of the set of frequent closed itemsets, we now define the closed itemsets.

1.4.2 Closed Itemsets

Closed itemsets are those that do not have any superset with the same frequency. They represent the maximal itemsets within equivalence classes.

Definition 1.2 (Closed itemset (Pasquier et al. [1999]))**.** An itemset P is *closed* if and only if there does not exist any itemset $Q \supsetneq P$ such that $freq(Q) = freq(P)$.

Example 1.6. The itemset BC is closed, but BE is not as $freq(BE) = freq(BCE) = 4$. The closed itemset of the equivalence class containing the itemsets D, CD, BD, BCD is the itemset BCD.

Definition 1.3 (Closure extension)**.** An non-empty itemset Y is a closure extension of X iff $cover(X \cup Y) = cover(X)$.

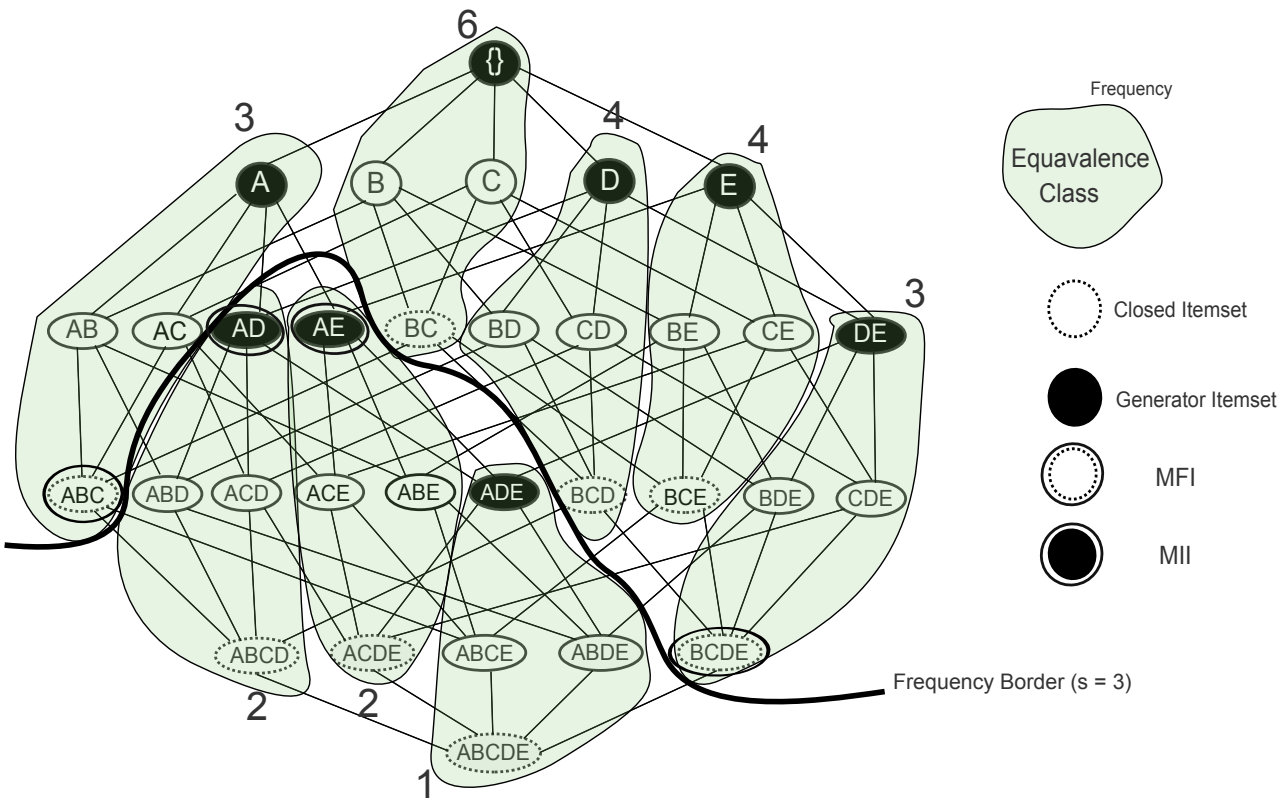


Figure 1.3 – *Equivalence classes of the dataset in Table 1.2a*

The closure extension is used for mining closed itemsets in (Wang et al. [2003]). In fact, X cannot be closed if it has a closure extension Y.

Frequent closed itemsets are those closed and frequent in addition. Usually, frequent itemsets mining is reduced to mining the closed frequent itemsets (Pasquier et al. [1999]). Several methods for mining closed frequent itemsets have been introduced, and LCM (Linear time Closed itemset Miner) (Uno et al. [2003, 2004, 2005]) is the most popular for its efficiency.

1.4.3 Minimal Infrequent Itemsets

Minimal infrequent itemsets represent the smallest set possible from which all infrequent itemsets are reproduced. It is usually significantly smaller than the set of infrequent itemsets. In (Mannila and Toivonen [1997]) MIIs are referred to as negative border.

Definition 1.4 (Minimal Infrequent Itemset (MII))**.** An itemset is an MII if it is infrequent and all its proper subsets are frequent.

Example 1.7. With $s = 3$, the dataset in Table 1.2a has MIIs = $\{AD^2, AE^2\}$. AD is infrequent ($freq(AD) = 2 < 3$) and all its proper subsets are frequent ($freq(A) = 3 \geq 3$ and $freq(D) = 4 \geq 3$).

Several techniques have been introduced for mining MIIs (Szathmary et al. [2007b], Haglin and Manning [2007], Szathmary et al. [2012]). It is obvious that the set of MIIs is a subset of the set of generator itemsets. Most techniques take advantage of this to mine MIIs.

1.4.4 Generator Itemsets

An itemset is a generator if there does not exists any subset with the same frequency. Generators were first introduced in (Bastide et al. [2000b]) for efficiently mining frequent itemsets. In (Szathmary et al. [2007a]) frequent generators are used for mining the *minimal non-redundant rules*. For efficiently mining MIIs, generators are used in (Szathmary et al. [2007b]).

Definition 1.5 (Generator (Bastide et al. [2000b]))**.** A *generator* is an itemset P such that there does not exist any itemset $Q \subsetneq P$ such that $freq(Q) = freq(P)$.

Example 1.8. The itemset DE is generator because $freq(DE) = 3$ and none of its subsets ($\emptyset$, D, E) have the same frequency ($freq(\emptyset) = 6$, $freq(D) = freq(E) = 4$). The itemset BE is not a generator because it has the same frequency as one of its subsets: $freq(BE) = freq(E) = 4$.

An interesting propriety on generators is that all supersets of a non-generator are non-generators too.

Proposition 1.2 (Bastide et al. [2000b]). Given two itemsets P and Q, if P is not a frequent generator and $P \subsetneq Q$, then Q is not a frequent generator either.

Proposition 1.2 holds on frequent generators but it could be generalized to all generators by simply setting the frequency threshold to 0. Proposition 1.2 is used in (Szathmary et al. [2009]) for efficiently mining frequent generators, the algorithm is called TALKY-G.

1.4.5 The Dualization Problem

It is obvious that a duality exists between MFIs and MIIs. The problem of inferring MFIs from the set of MIIs and vice versa is called the dualization. In (Mannila and Toivonen [1997], Boros et al. [2003]) and (Nourine and Petit [2012]) this problem is reduced to computing the minimal transversals of a hypergraph (Berge [1989]). In fact, MIIs correspond to the minimal transversals of the hypergraph constructed by the complements of the set of MFIs.

Given a hypergraph $H = (V, E)$, a transversal of H is a subset of V that has non empty intersection with every edge in E. A transversal is minimal if it has no subset which is a transversal. We denote by $Tr(H)$ the set of all minimal transversals of the hypergraph H. We know from (Mannila and Toivonen [1997]) that MIIs $= Tr(H)$, where H is a hypergaph

constructed by the complements of the set of MFIs. On the other hand, MFIs are the complements of $Tr(H)$ where H represents the set of MIIs.

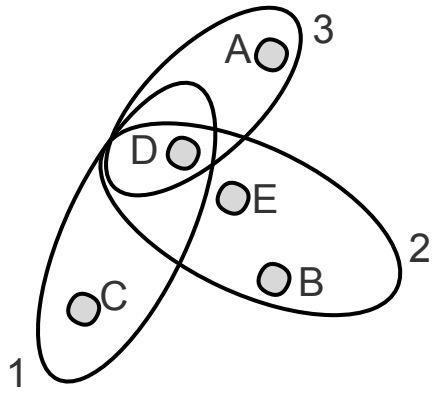


Figure 1.4 – *The hypergraph H (complements of MFIs)*

Example 1.9. With $s = 3$, the dataset in Table 1.1 has MFIs = $\{ABE^3, AC^3, BCE^3\}$ and MIIs = $\{ABC^2, ACE^2, D^1\}$. Let $H = (V, E)$ be a hypergraph represented in Figure 1.4. It is constructed using the complements of the set of MFIs (e.g., the complement of ABE is CD) where $V = \{A, B, C, D, E\}$ and $E = \{1, 2, 3\}$. It is clear that MIIs $= Tr(H) = \{ABC, ACE, D\}$.

1.5 Association Rules

Mining association rules aims at discovering interesting regularities between items in large-scale datasets. Association rules (ARs) were originally introduced by Agrawal et al. [1993] for sales transactions and products. An association rule captures an information of the kind *"if we have A and B, the chances to have C are high"*. Nowadays, a broad spectrum of application domains ask for this kind of information with a variety of datasets.

A common strategy used by many algorithms for mining association rules is to proceed in two steps. This is by first generating frequent itemsets then generate confident rules from the extracted frequent itemsets. The first step was presented in Section 1.3. Next, we present the common strategy used for the second step.

1.5.1 Rules Generation Step

Rules generation step extracts confident rules from frequent itemsets. Unlike the frequency, the confidence is neither anti-monotone nor monotone. A brute force strategy is practically infeasible. For a frequent itemset of size k there are about 2^k rules to test for confidence. Luckily, using the following proposition helps pruning the search space.

Proposition 1.3. If a rule $X \rightarrow Y \setminus X$ is not confident, then any rule $X' \rightarrow Y \setminus X'$ such that $X' \subsetneq X$, is not confident too.

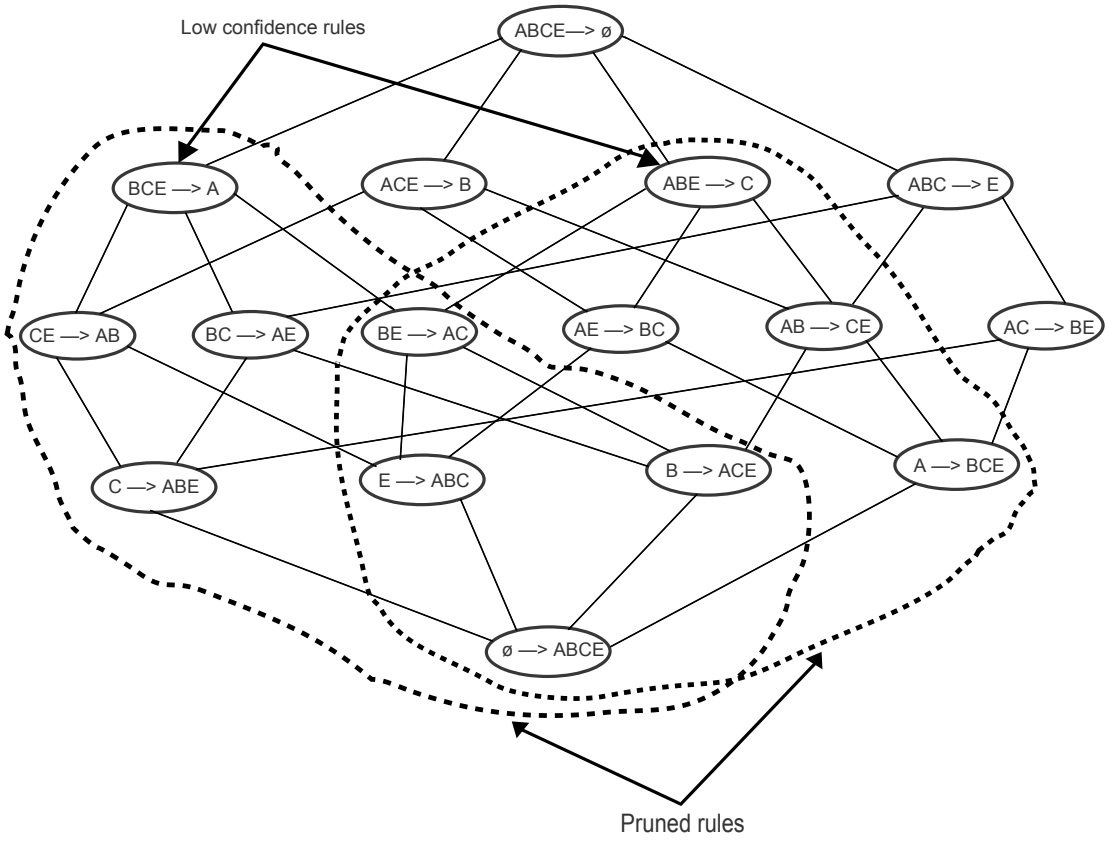


Figure 1.5 – *Pruning of association rules on the dataset in Table 1.1 with* $s = 2$ *and* $c = 70\%$.

Example 1.10. The Figure 1.5 illustrates the pruning presented in Proposition 1.3 on the dataset in Table 1.1. With $s = 2$ the itemset $ABCE$ is frequent. The lattice in Figure 1.5 presents all the possible rules using $ABCE$. When testing the confidence of the rule $BCE \rightarrow A$ we find that $conf(BCE \rightarrow A) = 66.66\%$. With $c = 70\%$ the rule $BCE \rightarrow A$ is not confident. As a result the rules $\{CE \rightarrow AB, BC \rightarrow AE, BE \rightarrow AC, C \rightarrow ABE, E \rightarrow ABC, B \rightarrow ACE, \emptyset \rightarrow ABCE\}$ are pruned using Proposition 1.3.

1.5.2 Condensed Representations for Association Rules

In practice, the number of valid rules is often huge and can easily exceed the size of the dataset itself. In this case the user faces an enormous number of irrelevant rules. However, the user may also ask for some *representative rules* according to a condensed representation. In this section, we present some well known condensed representations for association rules.

Representative rules

Representative association Rules (RRs) (Kryszkiewicz [1998a]) is a smaller set of rules that covers all rules. It uses a rule cover operator to derive all association rules. We now define the rule cover operator.

Definition 1.6 (Rule Cover Operator (Kryszkiewicz [1998a])). Given a rule $r : X \rightarrow Y$, the set of rules covered by r, denoted $C(r)$, is defined as $C(X \rightarrow Y) = \{X \cup Z \rightarrow V \mid Z, V \subseteq Y,\ Z \cap V = \emptyset,\ V \neq \emptyset\}$.

Example 1.11. The rule cover of $A \rightarrow BCD$ is $C(A \rightarrow BCD) = \{A \rightarrow BCD, A \rightarrow BC, A \rightarrow BD, A \rightarrow CD, A \rightarrow B, A \rightarrow C, A \rightarrow D, AB \rightarrow CD, AC \rightarrow BD, AD \rightarrow BC, AB \rightarrow C, AC \rightarrow B, AD \rightarrow B, AB \rightarrow D, AC \rightarrow D, AD \rightarrow C, ABC \rightarrow D, ABD \rightarrow C, ACD \rightarrow B\}$.

Mining the set of RRs allows the reproduction of all rules using the rule cover operator. Let $AR(s, c)$ be the set of association rules w.r.t. the thresholds s and c, and $RR(s, c)$ the corresponding representative rules.

Definition 1.7 (Representative Rules (Kryszkiewicz [1998a])). The set of RRs w.r.t. the thresholds s and c, denoted $RR(s, c)$, is $RR(s, c) = \{r \in AR(s, c) \mid \nexists r' \in AR(s, c)\ s.t\ r' \neq r,\ r \in C(r')\}$.

Example 1.12. In the dataset in Table 1.1, $RR(2, 50\%) = \{\emptyset \rightarrow AC, \emptyset \rightarrow ABE, \emptyset \rightarrow BCE, A \rightarrow BCE, B \rightarrow ACE, C \rightarrow ABE, E \rightarrow ABC\}$.

Minimum body maximum head

A stronger version of representative rules consists in looking for rules with minimum body and maximum head (Kryszkiewicz [1998b]). A rule is Minimum body Maximum head Rule (MMR) if it is valid and it has the minimum possible body and the maximum possible head.

Definition 1.8 (Minimum body Maximum head Rule (Kryszkiewicz [1998b])). The set of MMRs w.r.t. the thresholds s and c, denoted $MMR(s, c)$, is $MMR(s, c) = \{r : X \rightarrow Y \in AR(s, c) \mid \nexists r' : X' \rightarrow Y' \in AR(s, c)\ s.t\ r' \neq r,\ X' \subseteq X,\ Y' \supseteq Y\}$.

Example 1.13. In the dataset in Table 1.1, $MMR(2, 50\%) = \{\emptyset \rightarrow AC, \emptyset \rightarrow ABE, \emptyset \rightarrow BCE, B \rightarrow ACE, E \rightarrow ABC\}$.

It is clear that the set of MMRs is a subset of the set of RRs. However, the set of MMRs does not allow the reproduction of all valid association rules as shown in the following example.

Example 1.14. From the previous example, the valid rule $A \rightarrow BCE$ cannot be reproduced using the rules in $MMR(2, 50\%)$.

Minimal non-redundant association rules (MNRs)

The most common notion of representative rules are the *minimal non-redundant association rules* (MNR) (Bastide et al. [2000a]). A rule is an MNR if there does not exist any other rule with the same frequency and the same confidence that is obtained by removing items from the body or adding items to the head.

Definition 1.9 (Rule dominance)**.** We say that a rule $r : X \rightarrow Y$ dominates another rule $r' : X' \rightarrow Y'$ denoted by $r \succ r'$ iff $X \subseteq X'$, $Y \supseteq Y'$, $freq(r') = freq(r)$, $conf(r') = conf(r)$, and $r' \neq r$.

Example 1.15. In the dataset in Table 1.1, the rule $D \rightarrow ABE$ dominates the rule $ADE \rightarrow B$ because $D \subseteq ADE$, $ABE \supseteq B$ and $freq(ADE \rightarrow B) = freq(D \rightarrow ABE) = 1$ and $conf(ADE \rightarrow B) = conf(D \rightarrow ABE) = 100\%$.

Definition 1.10 (Minimal Non-redundant Rule (MNR) (Bastide et al. [2000a]))**.** Given a frequency threshold s and a confidence threshold c, a *non-redundant association rule* r is a valid rule where there does not exist any rule r' such that $r' \succ r$.

Note that any valid rule is whether an MNR or it is dominated by an MNR.

Example 1.16. In the dataset in Table 1.1 with $s = 1$ and $c = 100\%$ the set of MNRs is $\{E \rightarrow B, D \rightarrow ABE, CE \rightarrow B, B \rightarrow E, BC \rightarrow E, AE \rightarrow B, ACE \rightarrow B, AB \rightarrow E, ABC \rightarrow E\}$.

There exists an interesting operational characterization of MNRs.

Proposition 1.4 (Bastide et al. [2000a]). An association rule $X \rightarrow Y$ is an MNR if and only if:

$$\begin{cases} freq(X \rightarrow Y) \geq s \wedge conf(X \rightarrow Y) \geq c & (1) \\ X \text{ is a generator} & (2) \\ X \cup Y \text{ is closed} & (3) \end{cases}$$

Note that we can find a different kind of non-redunbdant rules in (Zaki [2004]) where we ask for rules of minimal body and minimal head.

1.5.3 Variants of Association Rules

In real word problems, the user may ask for a different kind of rules comparing to the classical definition. In the following, we review some variants of association rules.

Rare association rules

Rare association rules are those infrequent but confident. A rare rule captures items that rarely occur in the dataset but when they occur they occur together. To generate this kind of rules one should generate infrequent itemsets w.r.t. s then generate confident rules from those itemsets w.r.t. c.

The number of rare rules is usually huge making their enumeration infeasible. In (Szathmary et al. [2010]) mining rare rules is reduced to mining a set of informative rules to remove redundancy. This set is called Minimal Infrequent Generator Rules (MIGRs). In addition of being infrequent and confident, MIGRs represent a smallest set from which all rare rules can be reproduced.

Negative association rules

Negative association rules consider the absence of items in a transaction dataset (Wu et al. [2004]). Negative association rules are of the form $\neg X \rightarrow Y$, $X \rightarrow \neg Y$ or $\neg X \rightarrow \neg Y$, meaning that the absence of X implies the presence of Y, the presence of X implies the absence of Y and the absence of X implies the absence of Y, respectively.

Disjunctive association rules

A disjunctive association rule allows the disjunctive operator between itemsets (Nanavati et al. [2001]). Hence, we can have rules of the form $X \rightarrow Y \vee Z$ or $X \vee Y \rightarrow Z$. These kind of rules can express the following information for instance "People buying cake also buy coffee **or** tea".

Fuzzy association rules

Fuzzy association rules (Kuok et al. [1998]) are introduced to deal with quantitative attributes. Fuzzy rules are of the form "if X is A then Y is B" where X and Y are attributes and A and B are fuzzy sets of X and Y, respectively. Fuzzy sets represent qualitatively an interval of an attribute. For instance we can represent the *age* from 25 to 55 as *middle* or a *salary* from $4,000$ to $10,000$ as *high*. Then we can get the following kind of information "if *age* is *middle* then *salary* is *high*" which is more understandable for human.

Weighted association rules

In contrary to the traditional association rules mining, for weighted association rules the quantity of the item in a transaction is considered (Wang et al. [2000b]). For example, a customer may purchase 6 bottles of soda and 5 backs of snacks. In this case a weighted

association rule could be $soda[4,6] \rightarrow snack[3,5]$ meaning that customers buying between 4 and 6 sodas are more likely to buy between 3 and 5 snacks. This can help a manager of a supermarket to propose a promotion such as "if a customer buys 8 bottles of soda, he can get two free bags of snacks".

Indirect association rules

An indirect association rule captures items that rarely occur together but frequently occur with other items (Tan et al. [2000]). This can be used in some practical cases. For instance, in a supermarket transaction data, the items a and b may represent two products of competing companies A and B, respectively. The company B wants to convince people buying the product a to buy the product b instead. A way to identify these people is to look for those that rarely buy b, but may buy same items as people buying b. That is because both communities (people buying a and those buying b) are interested in the type of products produced by companies A and B.

1.5.4 ZART and ECLAT-Z for Mining Association Rules

ZART (Szathmary et al. [2007a]) and its closest brother ECLAT-Z (Szathmary et al. [2009]) are known to be of high efficiency methods for mining association rules and MNRs.

ZART is a multifunctional itemset mining algorithm. Its basic idea is to find a link between frequent generators and their closures. For this, ZART starts by mining frequent itemsets using the counting inference as in (Bastide et al. [2000b]). The counting inference allows the inferring of the frequency of non-generator itemsets without scanning the data. In fact when an itemset P is not a generator its frequency equals to the frequency of one of its subsets. We know that the frequency of an itemset is never larger than the frequency of its subset. As a result the frequency of a non-generator is the minimum among the frequencies of its subsets, i.e. $freq(P) = min_{i \in P}(freq(P \setminus \{i\}))$. ZART also identifies closed frequent itemsets. It starts by labelling all frequent itemsets with ″closed", then the label value changes to ″not closed" for frequent itemsets having a superset with the same frequency. Finally, ZART gathers frequent generators with their frequent closed itemset in a single equivalence class and represents the inclusion relation between equivalence classes with arrows (see Figure 1.6).

For mining association rules, ZART generates rules of the form $X \rightarrow Y$ where X is a frequent generator and $X \cup Y$ is a frequent closed itemset. Rules generated using generators and their proper closed itemsets are called *Generic Basis* with a confidence of 100%. *Informative Basis* are those generated using generators and closed itemsets of other equivalence classes. The set of MNRs is then the union of *Generic Basis* and *Informative Basis* rules.

Example 1.17. Figure 1.6 illustrates the result of ZART on the dataset in Table 1.1 with $s = 2$. The inclusion relation is represented by arrows between equivalence classes. The rule $AB \rightarrow E$ is *Generic Basis*, that is because it uses a closed itemset and a generator from

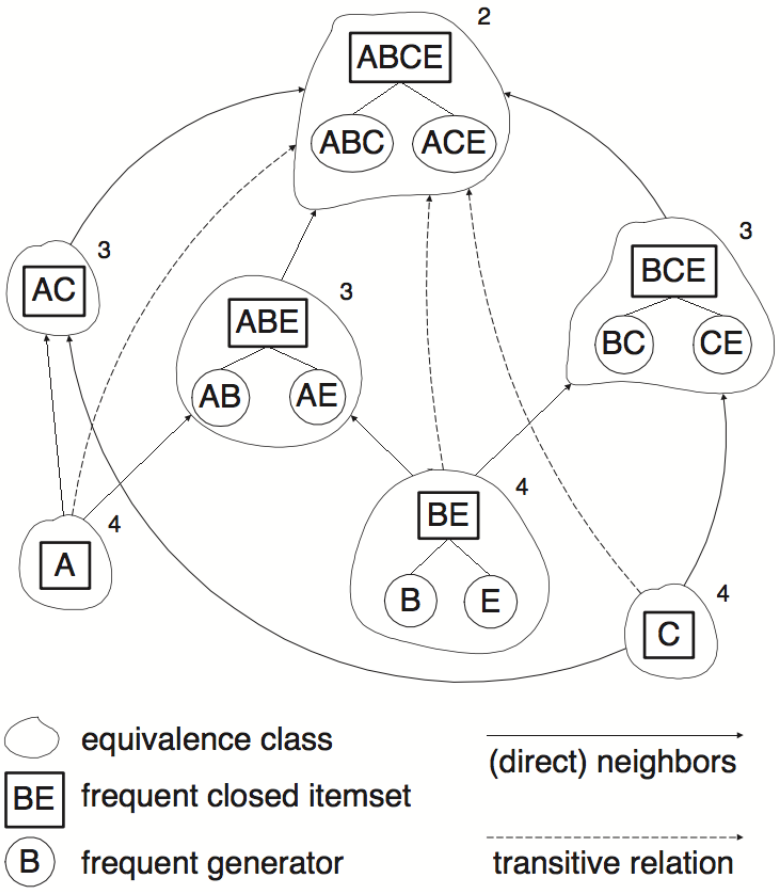


Figure 1.6 – *Result of* ZART *on the dataset in Table 1.1 with* $s = 2$ *(Szathmary et al. [2007a]).*

the same equivalent class for its generation. Its confidence is obviously 100%. The rule $A \rightarrow BE$, on the other hand, is *Informative Basis* with a confidence of $conf(A \rightarrow BE) = \frac{3}{4} = 75\%$.

ECLAT-Z is a hybrid way to reach the same result as ZART. ECLAT-Z is a wedding between a depth-first algorithm (e.g., FP-GROWTH) and a levelwise algorithm (e.g., APRIORI). ECLAT-Z uses a depth-first algorithm (i.e, ECLAT (Zaki [2000])) to mine frequent itemsets. Then it filters frequent closed and generator itemsets among frequent itemsets in a levelwise manner like ZART does.

1.5.5 Measures for Evaluating Association Rules

The frequency and the confidence are the most used measures for rules mining. However, in many real world problems they can be misleading. In this case more expressive measures are required. For instance, the rule $\{Soda\} \rightarrow \{Jus\}$ may be considered interesting for its high confidence of 80%. However, knowing that 90% of people are Jus drinkers decreases the probability of a person who drinks $Soda$ to drink Jus from 90% to 80%. The lift can avoid these kind of misleadings. The lift is the fraction of the confidence on the relative frequency of the head. More formally, $Lift(X \rightarrow Y) = \frac{conf(X \rightarrow Y)}{\frac{freq(Y)}{|\mathcal{T}|}}$. The lift represents

the degree of correlation between X and Y. If $Lift(X \rightarrow Y) = 1$, X and Y are independent, if $Lift(X \rightarrow Y) < 1$, X and Y are negatively correlated and if $Lift(X \rightarrow Y) > 1$, X and Y are positively correlated.

Example 1.18. The rule $A \rightarrow BE$ in Table 1.1 has a lift of $Lift(A \rightarrow BE) = \frac{conf(A \rightarrow BE)}{\frac{freq(BE)}{5}} = \frac{0.75}{0.8} = 0.94$. The lift being less than 1 means that the presence of A does not increase the chance of having BE (the probability to have BE is reduced from 80% to 75%).

A number of measures can be found in the literature for evaluating rules. Some are symmetric and others are asymmetric. Given a rule $X \rightarrow Y$ symmetric measures are those that reason on the itemset $X \cup Y$, items being in the body or in the head is not important for these measures, i.e. $Measure(X \rightarrow Y) = Measure(X \cup Y)$. An example of symmetric measures is the frequency. Asymmetric measures, on the other hand, do take in consideration the location of X and Y and $Measure(X \rightarrow Y)$ does not necessary equal $Measure(Y \rightarrow X)$. An example of asymmetric measures is the confidence.

1.6 Constrained Patterns

The interestingness of a pattern is evaluated through queries in inductive mining. The theory of a dataset is the set of interesting patterns, i.e. patterns that satisfy a given query (Mannila and Toivonen [1997]). For instance, mining frequent itemsets of at least a given size can be represented using a query as follows. Mine all itemsets P such that $freq(P) \geq s \wedge |P| \geq lb$, where s is the frequency threshold and lb is the lower-bound size of the itemset P.

The strategy followed by specialists to compute the set of satisfied patterns is highly influenced by the type of the query. The constraints can be monotone, anti-monotone, succinct (Ng et al. [1998]), or loose anti-monotone (Bonchi and Lucchese [2005]).

Designing a special strategy for every constraint can be tedious specially when combining traditional constraints with additional user's constraints. Such user's constraints are handled by data mining researchers, whenever possible, with (i) a pre-processing step reducing the dataset; (ii) a post-processing step to filter out the undesirable patterns; (iii) a filtering integrated in the specialized algorithm.

For mining association rules, Ng et al. [1998] propose an architecture that allows an interaction with the user. First, it generates itemsets according to the user's defined constraints/thresholds. If the user does not like the returned result it has the chance to change her constraints/thresholds. In this case, the process needs to be entirely revised to take in consideration the user's new constraints. Then, as soon as the user is satisfied, the process moves to the extraction of rules from the validated itemsets by, again, considering the user's constraints.

Next, we address the problem of mining condensed representation in the presence of additional queries (constraints).

1.6.1 Constrained Condensed Representations

As pointed out in (Bonchi and Lucchese [2004]), constraints can interfere with closeness (or maximality) when they are not monotone. Likewise, constraints can interfere with minimality when they are not anti-monotone. In fact constraints on maximal/minimal itemsets can be interpreted in two ways:

— I_1: mine maximal/minimal itemsets that, in addition, satisfy the constraints.

— I_2: mine itemsets that are maximal/minimal among those satisfying the constraints.

We are usually interested in mining itemsets w.r.t. the second interpretation.

For monotone constraints on the maximality (MFIs or closed itemsets) I_1 is equivalent to I_2. Likewise, I_1 is equivalent to I_2 for anti-monotone constraints on the minimality (MIIs or generators).

Definition 1.11 (A monotone constraint)**.** We call c a monotone constraint on itemsets if: $\forall P, Q \in \mathcal{I} \mid Q \supseteq P$: $c(P)$ is true $\implies$ $c(Q)$ is true.

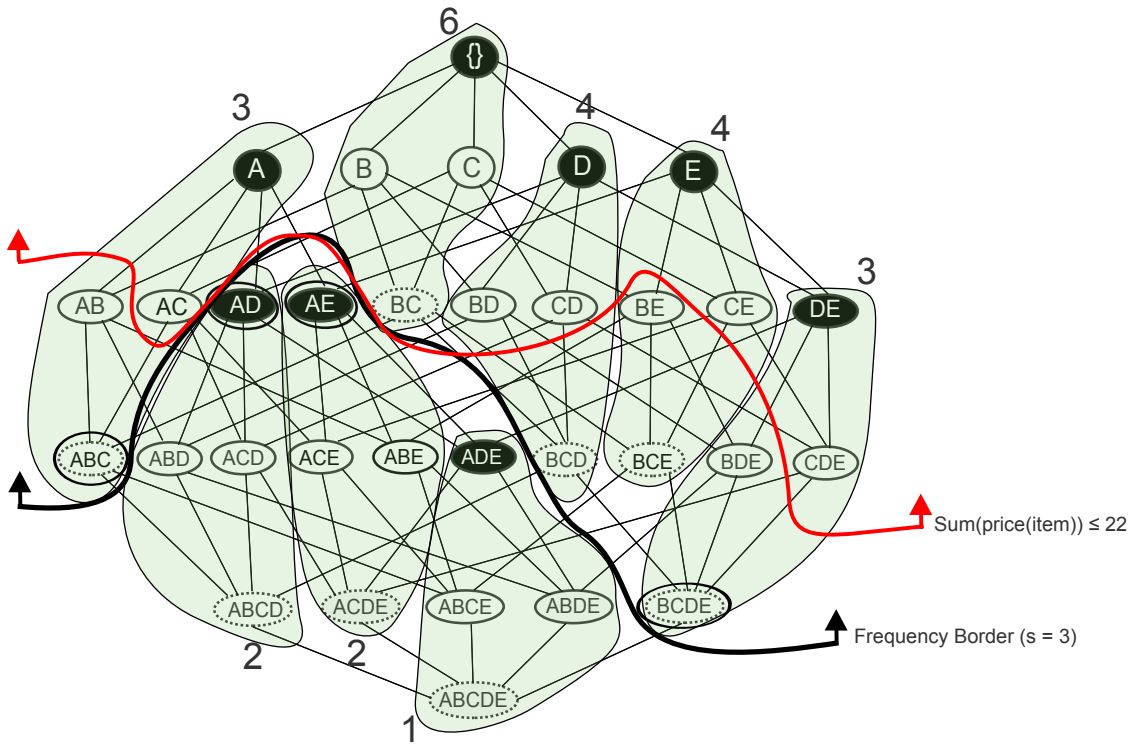
Definition 1.12 (An anti-monotone constraint)**.** We call c a anti-monotone constraint on itemsets if: $\forall P, Q \in \mathcal{I} \mid Q \subseteq P$: $c(P)$ is true $\implies$ $c(Q)$ is true.

Example 1.19. Figure 1.7 illustrates the set of itemsets from the dataset in Table 1.2a that satisfy an anti-monotone constraint 1.7(a) or a monotone constraint 1.7(b), in addition to the frequency ($s = 3$). The anti-monotone constraint c_{am} ensures that $\sum_{i \in P} price(i) \leq 22$ where P is an itemset. The monotone constraint c_m ensures that $\sum_{i \in P} price(i) \geq 33$. The set of frequent closed itemsets that in addition satisfy c_{am} are only BC, whereas the set of frequent closed itemsets among those satisfying c_{am} are AC, BC, BD, CD, CE and CDE. The set of frequent generators is the same for both interpretations (see Figure 1.7(a)). No frequent generator satisfies in addition c_m, whereas the set of frequent generators among those satisfying c_m are AB, BCE and BDE. The set of frequent closed itemsets is the same for both interpretations (see Figure 1.7(b)).

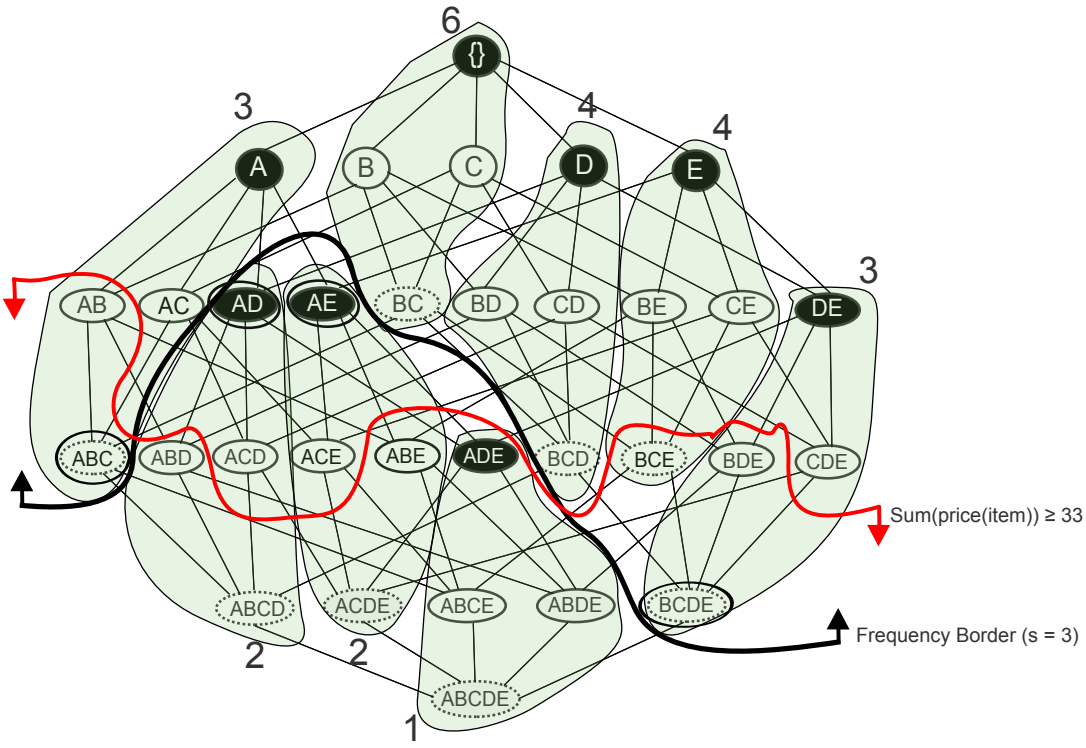
1.7 Data Compression

An entirely different strategy of itemset mining is not to extract a condensed representation of the set of frequent itemsets, but instead to look for frequent itemsets that yield to a compression that minimizes the lossles of the dataset (Siebes et al. [2006]). The KRIMP method (Vreeken et al. [2011]) uses MDL (Minimal Description Length) principle to find frequent itemsets that, together, describe the dataset best. For MDL the best set of patterns is the one that compresses the data best (Rissanen [1978], Grünwald [2004]). More formally, MDL selects the hypothesis H that minimizes $L(H) + L(D \mid H)$, where $L(H)$ is the length of H and $L(D \mid H)$ is the length of the data D when encoding with H.

For the KRIMP method the hypothesis is a two-columns table called the Code Table (CT), where the first column contains an itemset and the second one contains its associated code. KRIMP then uses some heuristics to find an approximate solution, i.e. CT, that minimizes $L(CT) + L(\mathcal{D} \mid CT)$.



(a)



(b)

Figure 1.7 – *Condensed representations with constraints*

1.8 Other Data Mining Problems

In this thesis, we focus on solving some itemset mining problems from datasets where transactions are sets of items. But it is worthy mentioning that other types of data exist in the literature. In this section we give a brief overview on some well known problems other than itemset mining problems.

1.8.1 Sequence Mining

In sequence mining the dimension of time is considered (Agrawal and Srikant [1995]). In this case, the order of items is important and repetition of symbols is allowed.

Let $\mathcal{I}$ be a set of items and $\mathcal{T}$ a set of transaction indices. A sequence S is an ordered list of items, i.e. $\langle S_1 \dots S_k \rangle$ where k is the sequence length. A sequence database SD is a multiset of transactions t_i, where t_i is a sequence and $i \in \mathcal{T}$. We say that a sequence $P = \langle P_1 \dots P_m \rangle$ is a sub-sequence of $S = \langle S_1 \dots S_n \rangle$, denoted by $P \preceq S$, if $m \leq n$ and if it exist integers $1 \leq k_1 \leq \dots \leq k_m \leq n$ such that $P_i = S_{k_i}$. The cover of a sequence P, denoted by $cover(P)$, is the set of transaction indices i for which $P \preceq t_i$, i.e. $cover(P) = \{i \in \mathcal{T} \mid P \preceq t_i\}$. The frequency of a sequence is the cardinality of its cover, i.e. $freq(P) = |cover(P)|$. Given a frequency threshold s the problem of mining sequence patterns consists in finding all sequence patterns P such that $freq(P) \geq s$.

Table 1.3 – *Sequence database*

$trans.$	Sequence
t_1	$ABCBC$
t_2	$BABCA$
t_3	$ACBB$
t_4	$CAABACB$
t_5	AC

Example 1.20. Table 1.3 represents a sequence database where $\mathcal{I} = \{A, B, C\}$ and $\mathcal{T} = \{1, 2, 3, 4, 5\}$. The sequence AC is a sub-sequence of $BABCA$. The cover of the sequence BCB is $cover(BCB) = \{1, 4\}$. Its frequency is $freq(BCB) = |cover(BCB)| = 2$. Given $s = 3$ the sequence AC is frequent ($freq(AC) \geq 3$) but BCB is not.

1.8.2 Mining from Uncertain Data

In some real world data some attributes suffer from an uncertainty. Take *symptoms* in a medical dataset for instance, their presence come from subjective observations. In uncertain datasets the presence of an item in a trasaction is associated with a probability (Chui et al. [2007]). The structure of the dataset is the same as in itemset mining. The only difference is the additional dimension, i.e. the probability $p(i, t)$ of the presence of an item i in a transaction t. In uncertain datasets, the frequency of an itemset is defined as follows.

Given a transaction dataset, the existential probability of an itemset P in a transaction t is $p(P,t) = \prod_{i \in P} p(i,t)$. The frequency of the itemset P is then $freq(P) = \sum_{t_i} p(P,t_i)$.

Table 1.4 – *An uncertain dataset with 5 items and 5 transactions*

trans.	Items			
t_1	A (0.8)	B (0.9)	D (1)	E (0.5)
t_2	A (0.6)	C (0.3)		
t_3	A (0.2)	B (0.4)	C (0.6)	E (0.5)
t_4		B (0.8)	C (0.7)	E (0.9)
t_5	A (0.9)	B (0.6)	C (0.6)	E (0.3)

Example 1.21. In Table 1.4 we present an uncertain dataset with 5 transactions and 5 items. The probability of presence of item A in transaction t_1 is $p(A,t_1) = 0.8$. the existential probability of AB in t_1 is $p(AB,t_1) = 0.8 \times 0.9 = 0.72$. Its frequency is $freq(AB) = 0.72 + 0.08 + 0.54 = 1.34$.

1.9 Conclusion

In this chapter we have presented some background in itemset mining. We have focused specially on presenting (i) existing methods for mining frequent itemsets and its condensed representations (ii) existing methods for mining association rules and its variants. We have shown that specialized algorithms suffer from the lack of generality when it comes to mining patterns with additional constraints.

In this thesis we propose to solve some of the presented problems using constraint programming. Constraint programming allows a generic modeling of various problems. The user in this case only needs to extend the model to take in consideration her additional constraints and no revision of the solving process is required. In the next chapter we present some background in constraint programming.

II

Chapter 2: Constraint Programming

"Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." [Eugene Freuder]

2.1 Introduction

Constraint Programming (CP) is a powerful declarative paradigm. The user specifies the problem and the computer solves it. Constraint programming is the art of writing problems as models and solving them by finding solutions. Constraint programming dates back in the sixties, when Golomb and Baumert [1965] have introduced "Backtracking Programming".

In this chapter we give a brief background on constraint programming. We start with some notations in Section 2.2. We describe the solving process conducted by constraint solvers in Section 2.3. Global constraints are defined in Section 2.4. Finally, we define the reified constraints in Section 2.5. Further details such as constraint optimisation problems, soft constraints, numeric CSPs, dynamic CSPs, can be found in the handbook of constraint programming (Rossi et al. [2006]).

2.2 Notations and Definitions

A CP model is a general parameterized description of a class of problems. An instance of a CP model is associated with a constraint network.

A constraint network, $N = (X, D, C)$, is specified by:

- a finite set of variables $X = \{x_1, \ldots, x_n\}$.
- a finite set of domains $D = \{dom(x_1), \ldots, dom(x_n)\}$, where $dom(x_i)$ is the finite set of possible values for x_i.
- a finite set of constraints $C = \{c_1, \ldots, c_m\}$ on X. A constraint $c_j \in C$ is a relation that specifies the allowed combinations of values for its variables $X(c_j)$. $|X(c_j)|$ is called the arity of c_j. c_j is called a binary constraint if $|X(c_j)| = 2$. A constraint network is called a binary network if for all $c_i \in C$, c_i is a binary constraint.

An assignment (aka, instantiation) on a set $Y \subseteq X$ of variables, denoted by $A[Y]$, is a mapping from variables in Y to values, and a valid assignment is an assignment where all values belong to the domain of their variables. A solution is an assignment on X, $A[X]$, satisfying all constraints.

The Constraint Satisfaction Problem (CSP) consists in deciding whether an instance of a CP model has solutions (or in finding a solution).

Example 2.1. Consider the following instance of a CP model.

$$\begin{array}{rcl} X & = & \{x_1, x_2, x_3\} \\ D & = & \{dom(x_1), dom(x_2), dom(x_3)\} \text{ where,} \\ & & dom(x_1) = \{1, 2\}, dom(x_2) = \{0, 1, 2, 3\}, \\ & & dom(x_3) = \{2, 3\} \\ C & = & \{c_1(x_1, x_2), c_2(x_1, x_2, x_3)\} \text{ where,} \\ & & c_1(x_1, x_2) : x_1 > x_2 \\ & & c_2(x_1, x_2, x_3) : x_1 + x_2 = x_3 \end{array}$$

This CSP admits the two solutions $(x_1 = 2, x_2 = 0, x_3 = 2)$ and $(x_1 = 2, x_2 = 1, x_3 = 3)$.

2.3 Constraint Programming Solver

A trivial way to solve a CSP is to test every assignment of X for satisfiability. Intuitively, this strategy can be infeasible in practice. Thus, the solving process in constraint programming relies on two strategies, inference and search. The inference eliminates values from D, the set of domains, by constraint propagation. The search, on the other hand, explores values in D. Constraint solvers use backtracking search to explore the search space of partial assignments. At each assignment, constraint propagation algorithms prune the search space by enforcing local consistency properties such as domain consistency.

2.3.1 Backtracking

The backtracking algorithm explores the domains of variables by incrementally selecting variables and instantiate them. At the root the assignment is empty, then variables are instantiated one by one until $A[var]$ is not valid then it backtracks or until $var = X$, i.e.

the assignment is complete. More formally, a node nd in a search tree is an assignment $\{x_1 = v_1, \ldots, x_i = v_i\}$. nd is extended by selecting a variable x_j and branch on it with a selected value v_k from $dom(x_j)$. In this case, nd is updated to $nd \cup \{x_j = v_k\}$.

Example 2.2. Consider the search tree of the CSP in Example 2.3 represented in Figure 2.1. The node nd corresponds to the assignment $\{x_1 = 1\}$. It is extended to $\{x_1 = 1, x_2 = 2\}$ by branching on x_2 with 2.

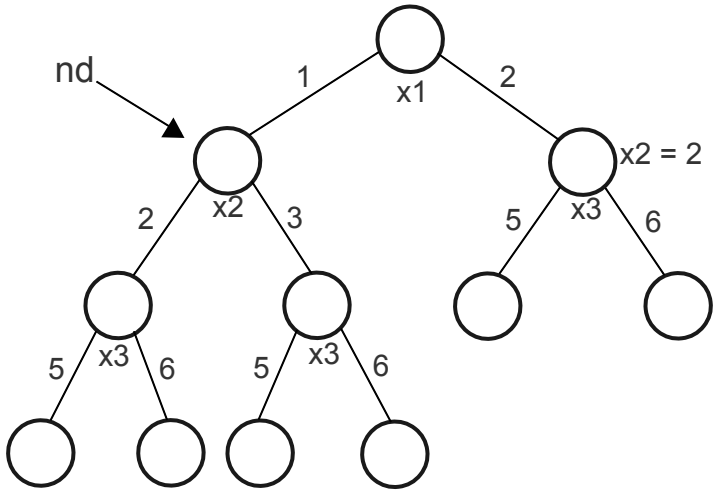


Figure 2.1 – *A search tree on Example 2.3*

Heuristics of variables/values selection. During the resolution, the algorithm of backtracking should select variables that are not instantiated to branch on them. It should also select a value from the domain of the selected variable to branch with. The order of the variable/value selection can highly influence the resolution time.

The variable selection strategy can be *static* or *dynamic*. For the static strategy, the order of variable selection is defined before the resolution starts (e.g., lexicographic order). For the dynamic strategy the order of the variable selection is defined during the resolution (e.g., select the variable with the smallest domain).

For the value selection strategy usually values of a given domain are selected in an increasing/decreasing order.

To make the backtracking efficient we can use:

- **Look back strategy** which learns from fails to avoid them in further search (Dechter [1990]).
- **Look ahead strategy** which anticipates fails by removing inconsistent values with an already instantiated variable $x_i = v_j$. This method is called Forward Checking and was first introduced under the name *preclusion* in (Golomb and Baumert [1965]).

To make the solving process even stronger, constraint propagation eliminates inconsistent values by enforcing some consistency properties such as domain consistency (aka,

arc consistency).

2.3.2 Propagation and Consistencies

Constraint propagation is a milestone in CP solvers. It is referred to with several names in the literature such as filtering algorithms. Constraint propagation reduces the search space by removing values from domains of some variables because a subset of constraints cannot be satisfied otherwise. For instance, consider the constraint $c : x_1 + x_2 \leq 5$, where domains of x_1 and x_2 are integers from 1 to 10. Clearly, the constraint c forbids values from 5 to 10 from both variables.

We say that a value v in $dom(x_i)$ has a support on the constraint c if there exists an assignment $A[X(c)]$ with $x_i = v$ such that c is satisfied. We now define a domain consistent constraint.

Definition 2.1 (Domain Consistency (DC))**.** A constraint c on $X(c)$ is domain consistent, if and only if, for every $x_i \in X(c)$ and every $v_j \in dom(x_i)$, v_j has a support on c.

Domain consistency helps filtering forbidden (i.e., inconsistent) values by removing values that have no support on the constraint. We say that a filtering algorithm A (or a propagator) ensures domain consistency on a constraint c if all values with no support on c have been removed by A.

Example 2.3. Consider the constraint $c(x_1, x_2, x_3) : x_1 + x_2 = x_3$ with $dom(x_1) = \{1, 2, 4\}$, $dom(x_2) = \{2, 3, 5\}$ and $dom(x_3) = \{3, 8, 9\}$. Using a domain consistency filtering, we get $dom(x_1) = \{1, 4\}$, $dom(x_2) = \{2, 5\}$ and $dom(x_3) = \{3, 9\}$. For instance, value 2 in $dom(x_1)$ is removed because summing-up 2 with any value in $dom(x_2)$ is not equal to any value in $dom(x_3)$.

Given a constraint network $N = (X, D, C)$, if all constraints in C are domain consistent, we say that the network N is domain consistent, but this does not mean that N has solutions. We show this in the following example.

Example 2.4. Consider the following constraint network, $N = (X, D, C)$.

$$
\begin{array}{rcl}
X & = & \{x_1, x_2, x_3\} \\
D & = & \{dom(x_1), dom(x_2), dom(x_3)\} \text{ where,} \\
 & & dom(x_1) = \{1, 2\}, dom(x_2) = \{1, 2\}, \\
 & & dom(x_3) = \{1, 2\} \\
C & = & \{c_1(x_1, x_2), c_2(x_1, x_3), c_3(x_2, x_3)\} \text{ where,} \\
 & & c_1(x_1, x_2) : x_1 \neq x_2 \\
 & & c_2(x_1, x_2) : x_1 \neq x_3 \\
 & & c_3(x_2, x_3) : x_2 \neq x_3
\end{array}
$$

N is domain consistent, the constraints c_1, c_2 and c_3 are domain consistent. Take the constraint $c_1 : x_1 \neq x_2$ for instance, values 1 and 2 in $dom(x_1)$ have supports in $dom(x_2)$, i.e. 2 and 1, respectively. But N has no solution.

AC3 (AC for Arc Consistency) is the most well known algorithm for ensuring domain consistency on a network (Mackworth [1977a]). It was originally proposed for binary networks, then extended for arbitrary networks, i.e. GAC3 (G for Generalized) (Mackworth [1977b]). GAC3 tests if a value v in $dom(x_i)$ has a support, if not it can be removed. To avoid useless tests, GAC3 uses a list where it stores only candidate pairs (x_i, c_j). GAC3 has a time complexity of $O(er^3d^{r+1})$ and a space complexity of $O(er)$ where $e = |C|$, r is the greatest arity among constraints in C and d is the cardinality of the largest domain.

We find in the literature many attempts to improve the complexity of GAC3. GAC4 reduces the time complexity to $O(erd^r)$ by storing additional information leading to a space complexity in $O(erd^r)$ (Mohr and Henderson [1986]). Bessiere [1994] proposes GAC6. GAC6 is a compromise between GAC3 and GAC4. It keeps the worst-case time complexity of GAC4, but it stops the search once a support is found as in GAC3. AC2001 improves the average complexity (Bessiere and Régin [2001]).

Domain consistency is the strongest and most used consistency for achieving an efficient filtering. A weak kind of consisitency is the *bound consistency* (Lhomme [1993]). Bound consistency is usually used in CSPs with continuous domains. Other consistencies can be found in the literature (Debruyne and Bessiere [2001]).

Finally, constraint solvers combine the backtracking with constraint propagation techniques such as GAC3 to reach a good performance in solving problems.

2.4 Global Constraints

Global constraints are constraints defined by a relation on a non-fixed number of variables. These constraints allow the solver to better capture the structure of the problem. Examples of global constraints are ALLDIFFERENT, REGULAR, AMONG, etc. For instance, the constraint ALLDIFFERENT$(x_1, \ldots, x_n)$ ensures that the variables from x_1 to x_n should take different values where n can be any number in $\mathbb{N}^* \setminus \{1\}$.

Example 2.5. The CSP in Example 2.4 can be represented using the global constraint ALLDIFFERENT. ALLDIFFERENT(x_1, x_2, x_3) ensures that x_1, x_2 and x_3 should take different values. ALLDIFFERENT captures better the structure of the problem and detects that the problem is inconsistent and no solution exists.

The generic DC algorithms such as GAC3 and GAC6 do not provide global constraints with an efficient filtering. However, when the decomposition of a global constraint is Berge-acyclic (Beeri et al. [1983]) (such as REGULAR) it preserves DC. On such constraints, the generic algorithms perform well and are optimal. For other global constraints, it is not possible to efficiently propagate them by generic DC algorithms because these algorithms

are exponential in the number of variables of the constraint. Fortunately, for many global constraints (such as ALLDIFFERENT), dedicated propagation algorithms can be designed to achieve DC in time polynomial in the size of the input, that is, the domains and the required extra parameters.

For ALLDIFFERENT, Régin [1994] proposes a polynomial filtering algorithm ensuring DC. It uses the bipartite graph (the graph that links variables to their domains) to detect inconsistency. A *matching* in a graph is a subset of edges with no vertex in common. If the largest possible matching contains every vertex in $X(c)$ then the constraint is consistent and at least a solution exists.

2.5 Reified Constraints

In constraint programming, it exists a type of meta-constraint that associate a constraint with a truth variable to represent its satisfiability. A reified constraint $b \iff c$ associates a Boolean variable b to the truth value of the constraint c (Apt [2003]). The variable b is set to $true$ ($b = 1$) if the constraint c is satisfied, to $false$ ($b = 0$) otherwise.

The filtering conducted by such constraint is as follows:

- If the constraint c is satisfied then $b = 1$.
- If the constraint c is violated then $b = 0$.
- If $b = 1$ then the constraint c should be satisfied.
- If $b = 0$ then the constraint c should be violated.

Reified constraints are used to model several problems. It can be used to express that a number of constraints should be satisfied.

Example 2.6. Consider the following model with reified constraints.

$$b_1 \iff c_1 \qquad (1)$$

$$b_2 \iff c_2 \qquad (2)$$

$$b_1 + b_2 \geq 1 \qquad (3)$$

This model expresses that at least a constraint (c_1 or c_2) should be satisfied. Constraints (1) and (2) are reified constraints and constraint (3) ensures that at least b_1 or b_2 is set to 1.

2.6 Conclusion

In this chapter we have presented some background in constraint programming, We have shown how constraint solvers proceed in solving problems using backtracking and constraint propagation. We also have presented some type of constraints such as global constraints and reified constraints.

Being expressive and flexible, constraint programming is used to solve data mining problems. In the next chapter, we review some existing declarative approaches in itemset mining.

III

Chapter 3: Declarative Itemset Mining

3.1 Introduction

In a recent line of work, declarative [1] approaches are used to solve various data mining problems. We can find declarative approaches for itemset mining (Raedt et al. [2008], Khiari et al. [2010], Boudane et al. [2016], Lazaar et al. [2016], Schaus et al. [2017], Bessiere et al. [2018]), for sequence mining (Coquery et al. [2012], Kemmar et al. [2015], Aoga et al. [2017]), for graph mining (Kemmar et al. [2018]), for mining from uncertain datasets (Dlala et al. [2016]), for clustering (Dao et al. [2017], Chabert and Solnon [2017]). In terms of CPU time, declarative methods have not yet competed with ad hoc algorithms. However, their flexibility allows the modeling of complex user queries without revising the solving process.

In this chapter we review existing declarative approaches for itemset mining problems. We start by defining some notations in Section 3.2. We introduce the basic constraint programming model for itemset mining in Section 3.3. The global constraints ClosedPattern and CoverSize, constraints that are used in our models, are presented in Sections 3.4 and 3.5, respectively. Finally, we review some SAT models in itemset mining in Section 3.6.

3.2 Notations and Definitions

Most declarative methods use Boolean variables for representing patterns, where x_i represents the presence of the item $i \in \mathcal{I}$ in the pattern. x is a vector of Boolean variables, $(x_1, \ldots, x_{|\mathcal{I}|})$ to represent the solution.

1. In this manuscript we focus on declarative approaches that are based on constraints (CP and SAT).

Given a vector of Boolean variables x, we use the following notations:
— $x^{-1}(1) = \{i \in \mathcal{I} \mid dom(x_i) = \{1\}\}$.
— $x^{-1}(0) = \{i \in \mathcal{I} \mid dom(x_i) = \{0\}\}$.
— $x^{-1}(*) = \{i \in \mathcal{I} \mid dom(x_i) = \{0, 1\}\}$.

Note that $x^{-1}(1) \cup x^{-1}(0) \cup x^{-1}(*) = \mathcal{I}$ and $x^{-1}(1) \cap x^{-1}(0) \cap x^{-1}(*) = \emptyset$.

Example 3.1. Given a dataset with 4 items, i.e. $\mathcal{I} = \{A, B, C, D\}$, the instantiation $x = [1, 0, 1, 0/1]$ corresponds to the sets $x^{-1}(1) = \{A, C\}$, $x^{-1}(0) = \{B\}$ and $x^{-1}(*) = \{D\}$.

Given an integer variable p, we use the following notations:
— $UB(p)$ is the upper-bound value of p.
— $LB(p)$ is the lower-bound value of p.

Example 3.2. Given an integer variable p with $dom(p) = \{1, 2, 3\}$, its lower-bound is 1, i.e. $LB(p) = 1$ and its upper-bound is 3, i.e. $UB(p) = 3$.

3.3 Basic Constraint Programming Model for Itemset Mining

The first constraint programming model for itemset mining was intoduced in (Raedt et al. [2008]). It is based on reified constraints to connect item variables to transaction variables.

Variables. For this model, a Boolean variable x_i is associated to every item i to represent the presence of the item i in the returned itemset. A transaction t_j is associated with a Boolean variable y_j to represent if the returned itemset is in the transaction t_j or not.

Constraints. We now present the constraints. We use D_{ji} to represent the presence of the item i in the transaction t_j (see the binary representation of the dataset in Table 1.1).

$$\forall j \in \mathcal{T} : y_j = 1 \leftrightarrow \sum_{i \in \mathcal{I}} x_i(1 - D_{ji}) = 0 \qquad (1)$$

The constraint (1) encodes the cover of an itemset. It ensures that the variable y_j is set to 0 if the itemset $x^{-1}(1)$ is absent from t_j. That is, if only an item $i \in x^{-1}(1)$ is absent from t_j then $x_i(1 - D_{ji}) = 1$ and the constraint $\sum_{i \in \mathcal{I}} x_i(1 - D_{ji}) = 0$ is violated (i.e., $y_j = 0$).

$$\sum_{j \in \mathcal{T}} y_j \geq s \qquad (2)$$

The constraint (2) ensures that the itemset $x^{-1}(1)$ is frequent w.r.t. the threshold s.

The model that combines the constraints (1) and (2) allows the mining of all frequent itemsets. This model can be extended with other constraints to express other properties. The closeness, for instance, can be encoded using the following constraint.

$$\forall i \in \mathcal{I} : x_i = 1 \leftrightarrow \sum_{j \in \mathcal{T}} y_j(1 - D_{ji}) = 0 \qquad (3)$$

The constraint (3) ensures that the itemset $x^{-1}(1)$ is closed. If the item i is present in all transactions t_j such that $y_j = 1$ then x_i is set to one.

The maximality is encoded with the following constraint.

$$\forall i \in \mathcal{I} : x_i = 1 \leftrightarrow \sum_{j \in \mathcal{T}} y_j D_{ji} \geq s \qquad (4)$$

The constraint (4) ensures that the itemset $x^{-1}(1)$ is maximal. If the itemset $x^{-1}(1) \cup \{i\}$ is frequent we include item i in the itemset (i.e., $x_i = 1$).

This model has some limits in terms of memory and scalability. It requires an important number of variables/constraints to encode only the frequency, i.e. $(n + m)$ variables and $(n + m)$ constraints where n is the number of items and m is the number of transactions. On huge datasets the process of mining frequent itemsets became infeasible. Take for instance the dataset `Retail` [2] which has $16,470$ items and $88,162$ transactions, it requires more than $100,000$ variables and $100,000$ constraints to encode the problem of mining frequent itemsets using the reified constraints. To ensure the closeness $16,470$ additional constraints are required and $16,470$ more constraints are required to ensure the maximality. That is why it may be interesting to capture some properties using global constraints.

3.4 The Global Constraint CLOSEDPATTERN

The first global constraint proposed for itemset mining, CLOSEDPATTERN, was introduced in (Lazaar et al. [2016]) for mining closed frequent itemsets. A global constraint provides a better view of the structure of the problem to the solver. Lazaar et al. [2016] propose a complete filtering algorithm for CLOSEDPATTERN.

Definition 3.1 (CLOSEDPATTERN)**.** Let x be a vector of Boolean variables, s a support threshold and $\mathcal{D}$ a dataset. The global constraint CLOSEDPATTERN$_{\mathcal{D},s}(x)$ holds if and only if $x^{-1}(1)$ is a closed frequent itemset w.r.t. the threshold s.

Example 3.3. On the dataset in Table 1.1 and with $s = 3$, given the instantiation $x = [1, 0, 0, 1, 0]$ we have $x^{-1}(1) = \{A, D\}$ and $x^{-1}(0) = \{B, C, E\}$. The constraint CLOSEDPATTERN$_{\mathcal{D},3}(x)$ is not satisfied ($freq(AD) = 1 < 3$). With $x = [0, 1, 0, 0, 1]$ the constraint CLOSEDPATTERN$_{\mathcal{D},3}(x)$ is satisfied, because $x^{-1}(1)$, i.e. BE is frequent and closed.

2. fimi.ua.ac.be/data/

3.4.1 Propagator of CLOSEDPATTERN

In the following, we present the filtering rules of CLOSEDPATTERN. The first rule filters 0 from $dom(x_i)$ if $\{i\}$ is a closure extension of $x^{-1}(1)$ (see Definition 1.3). The second rule filters 1 from $dom(x_i)$ if the itemset $x^{-1}(1) \cup \{i\}$ is infrequent w.r.t. s. Finally, the third rule filters 1 from $dom(x_i)$ if $cover(x^{-1}(1) \cup \{i\})$ is a subset of $cover(x^{-1}(1) \cup \{j\})$ where j is an absent item, i.e. $j \in x^{-1}(0)$.

1. If $|cover(x^{-1}(1) \cup \{i\})| = |cover(x^{-1}(1))| \Rightarrow 0 \notin dom(x_i)$.
2. If $|cover(x^{-1}(1) \cup \{i\})| < s \Rightarrow 1 \notin dom(x_i)$.
3. If $cover(x^{-1}(1) \cup \{i\}) \subseteq cover(x^{-1}(1) \cup \{j\}) \wedge j \in x^{-1}(0) \Rightarrow 1 \notin dom(x_i)$.

Example 3.4. On the dataset in Table 1.1 and with $s = 2$, consider the partial assignment $x^{-1}(1) = \{B\}, x^{-1}(0) = \{A\}$ and $x^{-1}(*) = \{C, D, E\}$. Thanks to rule 1, 0 is filtered from $dom(x_E)$, because $|cover(BE)| = |cover(B)|$. Then, the rule 2 filters 1 from $dom(x_D)$ because $freq(BDE) = 1 < 2$. The value 1 can be filtered from $dom(x_D)$ by the rule 3 also. That is because $cover(BDE) \subseteq cover(ABE)$ and A is in $x^{-1}(0)$.

Complexity. The algorithm implementing the three filtering rules of CLOSEDPATTERN has a time complexity in $O(n^2 \times m)$ and a space complexity in $O(n \times m)$ where n is the number of items and m is the number of transactions.

It is proven in (Lazaar et al. [2016]) that the filtering rules 1, 2 and 3 enforce domain consistency on the constraint CLOSEDPATTERN.

Lazaar et al. [2016] propose a weak propagator for CLOSEDPATTERN by omitting the expensive quadratic filtering rule, i.e. rule 3. Using this propagator speeds-up the solving process in some cases despite its incompleteness.

3.5 The Global Constraint COVERSIZE

The global constraint COVERSIZE is introduced for computing the exact size of the cover of an itemset (Schaus et al. [2017]). This offers more flexibility in modeling problems. One can mine frequent (infrequent) itemsets by forcing the size of the cover to be greater or equal (less than) a frequency threshold.

Definition 3.2 (COVERSIZE)**.** Let x be a vector of Boolean variables, p an integer variable and $\mathcal{D}$ a dataset. The global constraint COVERSIZE$_{\mathcal{D}}(x, p)$ holds if and only if $p = |cover(x^{-1}(1))|$.

Example 3.5. On the dataset in Table 1.1, given the instantiation $x = [0, 0, 1, 0, 1]$ and $p = 3$ the constraint COVERSIZE$_{\mathcal{D}}(x, p)$ is satisfied because the frequency of $x^{-1}(1)$, i.e. CE equals p, i.e. 3. The instantiation $x = [1, 1, 1, 0, 0]$ and $p = 3$, on the other hand, does not satisfy COVERSIZE because $freq(ABC) = 2 \neq p$.

Domain consistency on the constraint COVERSIZE is proven to be NP-hard (Schaus et al. [2017]). As a result, the proposed propagator is not complete. We present next the propagator of the constraint COVERSIZE.

3.5.1 Propagator of COVERSIZE

In the following, we present the filtering rules of COVERSIZE. The first two rules update the bounds of p. The third rule filters 1 from $dom(x_i)$ if including the item i would result a cover size that is bellow $LB(p)$. Finally, the fourth rule filters 0 from $dom(x_i)$ if excluding the item i would result the increasing of $UB(p)$ knowing that $UB(p) = |cover(x^{-1}(1) \cup x^{-1}(*))|$.

1. If $|cover(x^{-1}(1))| < UB(p) \Rightarrow UB(p) \leq |cover(x^{-1}(1))|$.
2. If $|cover(x^{-1}(1) \cup x^{-1}(*))| > LB(p) \Rightarrow LB(p) \geq |cover(x^{-1}(1) \cup x^{-1}(*))|$.
3. If $|cover(x^{-1}(1) \cup \{i\})| < LB(p) \Rightarrow 1 \notin dom(x_i)$.
4. If $UB(p) = |cover(x^{-1}(1) \cup x^{-1}(*))| \wedge |cover(x^{-1}(1) \cup x^{-1}(*))| < |cover(x^{-1}(1) \cup x^{-1}(*) \setminus \{i\})| \Rightarrow 0 \notin dom(x_i)$.

Example 3.6. On the dataset in Table 1.1, consider the partial assignment $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{D\}$ and $x^{-1}(*) = \{A, C, E\}$ and $dom(p) = \{0, 1, 2, 3, 4, 5\}$, i.e. $LB(p) = 0$ and $UB(p) = 5$. The rule 1 updates the upper-bound of p to $|cover(B)|$, i.e. $UB(p) = 4$. The rule 2 updates the lower-bound of p to $|cover(ABCE)|$, i.e. $LB(p) = 2$. Given the partial assignment $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{C\}$, $x^{-1}(*) = \{A, D, E\}$, $UB(p) = 4$ and $LB(p) = 2$, the rule 3 filters 1 from $dom(x_D)$ because $|cover(BD)| = 1 < LB(p)$. Given the partial assignment $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{C\}$, $x^{-1}(*) = \{A, D, E\}$, $UB(p) = 2$ and $LB(p) = 2$, the rule 4 filters 0 from $dom(x_D)$ because $UB(p) = freq(ABDE) = 2$ and $freq(ABE) = 3 > freq(ABDE)$.

To enforce the frequency one can simply add the constraint $p \geq s$ and to enforce the infrequency one can add the constraint $p < s$. The propagator of COVERSIZE enforces DC on the frequency, but it does not on the infrequency. We can show this in an example.

Table 3.1 – *Dataset*

trans.	Items
t_1	$A\ B$
t_2	$A\quad C$
t_3	$\ \ B$
t_4	$A\ B\ C$

Example 3.7. On the dataset in Table 3.1 we want to mine infrequent itemsets with $s = 3$. We can express this by the following model: $\text{COVERSIZE}_{\mathcal{D}}(x, p) \wedge p < 3$. The constraint $p < 3$ updates the upper-bound of p to 2, i.e. $UB(p) = 2$. With the partial instantiation $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{A\}$ and $x^{-1}(*) = \{C\}$ we have: $UB(p) = 2$ and $LB(p) = |cover(BC)| = 1$. No filtering is conducted in this node, but value 0 for $dom(x_C)$ is not domain consistent because with $x_C = 0$ we have the complete instantiation $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{A, C\}$ and $freq(B) = 3 \notin dom(p)$.

To efficiently compute the cover of an itemset, a new data-structure was introduced in (Schaus et al. [2017]) called *ReversibleSparseBitset*. *ReversibleSparseBitset* allows the ignoring of zero words in the cover computation.

As opposed to CLOSEDPATTERN, the closeness is encoded in (Schaus et al. [2017]) with an independent global constraint, COVERCLOSURE. COVERCLOSURE$_{\mathcal{D}}(x)$ ensures that the itemset $x^{-1}(1)$ is closed and not necessary frequent. It uses the same filtering rules as in (Lazaar et al. [2016]) (rules 1 and 3).

3.6 SAT for Itemset Mining

A SAT problem (for SATisfiability) consists in deciding whether a formula in Conjunctive Normal Form (CNF) admits a solution or not. A formula is in CNF if it is written as a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of Boolean variables (aka, literals).

SAT is extensively used in itemset mining. It is used for detecting symmetries (Jabbour et al. [2013a]), for top-k itemsets mining (Jabbour et al. [2013b]), for association rules mining (Boudane et al. [2016, 2017]), for closed frequent itemets mining (Dlala et al. [2018]) and for maximal frequent itemsets mining (Jabbour et al. [2018]).

3.6.1 SAT for Mining Maximal Frequent Itemsets

Jabbour et al. [2018] propose a SAT approach for mining MFIs. Jabbour et al. [2018] extend a model for mining closed itemsets to mine MFIs (as an MFI is a closed itemset).

Variables. To model the SAT approach for mining MFIs two vectors of literals are required, x_i to represent the presence of the item i in the itemset, and p_j to represent the presence of the itemset $x^{-1}(1)$ in the transaction t_j.

Constraints. We now present the set of constraints (aka, formulas).

$$\bigwedge_{j \in \mathcal{T}} (\neg p_j \leftrightarrow \bigvee_{i \in \mathcal{I} \setminus t_j} x_i) \qquad (1)$$

The formula (1) encodes the cover of $x^{-1}(1)$. It ensures that p_j is set to *false* if it exists at least an item in $x^{-1}(1)$ that is not in t_j.

$$\sum_{j \in \mathcal{T}} p_i \geq s \qquad (2)$$

The constraint (2) ensures that $x^{-1}(1)$ is frequent w.r.t. s. The constraint (2) corresponds

to a cardinality constraint. To handle this kind of constraints in SAT, a transformation to CNF is required. In the literature, several researchers have addressed this problem (Warners [1998], Asín et al. [2011], Jabbour et al. [2014]). The challenge is to achieve a good size of the CNF formula and to keep a good propagation on the constraint after transformation.

$$\bigwedge_{i \in \mathcal{I}} ((\bigvee_{i \notin t_j} p_j) \vee x_i) \qquad (3)$$

The formula (3) ensures that the itemset $x^{-1}(1)$ is closed. It encodes the fact that if $cover(x^{-1}(1)) = cover(x^{-1}(1) \cup \{i\})$ then the item i should be in $x^{-1}(1)$ (i.e., x_i should be set to $true$).

To capture the maximality, one can add the following constraint.

$$\bigvee_{i \in \mathcal{I}} \neg x_i \rightarrow (\sum_{j \in \mathcal{T} | i \in t_j} y_j < s) \qquad (4)$$

The constraint (4) is a cardinality constraint and needs to be transformed to CNF. This transformation can generate a huge number of clauses making the mining of MFIs infeasible. That is why Jabbour et al. [2018] use blocking clauses instead. Blocking clauses are clauses added during the search to block the solver from going further and lead it to another level of the search tree.

The solver starts by setting item variables to $true$ whenever $x^{-1}(1)$ is frequent. Every time a solution is found, the blocking clause (5) is added to find MFIs that include other items comparing to the previous solution.

$$\bigvee_{i \in \mathcal{I} \setminus x^{-1}(1)} x_i \qquad (5)$$

Note that this approach is sensitive to the value selection strategy. That is, any changes in the strategy would interfere with the soundness of the model. As a result, this approach cannot take additional constraints into consideration during the resolution.

3.6.2 SAT for Association Rules

Boudane et al. [2016] propose a SAT-CP hybrid approach for mining association rules. It encodes some constraints in CNF and uses constraint propagation to handle cardinality constraints. To show its flexibility, this SAT-CP model is extended to capture indirect rules. Using the same model, Boudane et al. [2017] capture the set of MNRs. Next, we present the SAT-CP model for mining association rules.

Variables. To model the SAT-CP approach for mining association rules four vectors of literals are required: x_i to represent the presence of the item i in the body; y_i to represent the presence of the item i in the head; p_j to represent the presence of the itemset $x^{-1}(1)$ in the transaction t_j; and q_j to represent the presence of the itemset $x^{-1}(1) \cup y^{-1}(1)$ in the transaction t_j.

Constraints. We now present the set of constraints (aka, formulas).

$$(\bigvee_{i \in \mathcal{I}} x_i) \wedge (\bigvee_{i \in \mathcal{I}} y_i) \qquad (1)$$

The formula (1) ensures that $x^{-1}(1)$ and $y^{-1}(1)$ are not empty (note that the SAT approach does not allow the body to be empty).

$$\bigwedge_{i \in \mathcal{I}} (\neg x_i \vee \neg y_i) \qquad (2)$$

The formula (2) ensures that $x^{-1}(1) \cap y^{-1}(1) = \emptyset$.

$$\bigwedge_{j \in \mathcal{T}} (\neg p_j \leftrightarrow \bigvee_{i \in \mathcal{I} \setminus t_j} x_i) \qquad (3)$$

The formula (3) is similar to formula (1) from Section 3.6.1. It encodes the cover of $x^{-1}(1)$.

$$\bigwedge_{j \in \mathcal{T}} (\neg q_j \leftrightarrow \neg p_j \vee (\bigvee_{i \in \mathcal{I} \setminus t_j} y_i)) \qquad (4)$$

The formula (4) encodes the cover of $x^{-1}(1) \cup y^{-1}(1)$. It ensures that q_j is set to *false* if p_j is set to *false* ($x^{-1}(1)$ absent from t_j) or at least an item in $y^{-1}(1)$ is absent from t_j.

$$\sum_{j \in \mathcal{T}} q_j \geq s \qquad (5)$$

The constraint (5) is similar to formula (2) from Section 3.6.1. It ensures that $x^{-1}(1) \cup y^{-1}(1)$ is frequent w.r.t. s.

$$\frac{\sum_{j \in \mathcal{T}} q_j}{\sum_{j \in \mathcal{T}} p_j} \geq c \qquad (6)$$

Finally, the constraint (6) ensures that the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ is confident w.r.t. c.

The solutions to the SAT-CP model that combines the constraints from (1) to (6) are the set of association rules. It encodes the constraints from (1) to (4) in CNF and uses constraint propagation for the constraints (5) and (6).

Despite the cardinality part of the SAT-CP model, the size of the encoding of the formulas from (1) to (4) may be huge. With n the number of items and m the number of transactions, this approach requires $(2n + 2m)$ literals and more than $(2nm)$ clauses to

encode the formulas from (1) to (4). This makes the charging of the CNF to the RAM infeasible on some datasets. For instance, the formulas from (1) to (4) on the dataset `Retail` is encoded in a CNF file of size 63Gb.

SAT for MNRs

Boudane et al. [2017] propose to extend the SAT-CP model for association rules to mine only MNRs (see Definition 1.10). For this, it is sufficient to force $x^{-1}(1) \cup y^{-1}(1)$ to be closed and $x^{-1}(1)$ to be a generator (see Proposition 1.4). The previously presented model in extended using the following formulas.

$$\bigwedge_{i \in \mathcal{I}} ((\bigwedge_{j \in \mathcal{T}} q_j \rightarrow i \in t_j) \rightarrow x_i \vee y_i) \qquad (7)$$

The formula (7) is similar to formula (3) from Section 3.6.1. It ensures that the itemset $x^{-1}(1) \cup y^{-1}(1)$ is closed. It encodes the fact that if $cover(x^{-1}(1) \cup y^{-1}(1)) = cover(x^{-1}(1) \cup y^{-1}(1) \cup \{i\})$ then the item i should be in $x^{-1}(1) \cup y^{-1}(1)$ (i.e., either x_i or y_i should be set to $true$).

$$(\bigwedge_{i \in \mathcal{I}} x_i \rightarrow \bigvee_{(j \in \mathcal{T}, i \notin t_j)} (\bigwedge_{k \notin t_j \cup \{i\}} \neg x_k)) \vee (\sum_{k \in \mathcal{I}} x_k = 1) \qquad (8)$$

The formula (8) ensures that the itemset $x^{-1}(1)$ is generator. It represents the fact that if $freq(x^{-1}(1) \rightarrow y^{-1}(1)) = freq(x^{-1}(1) \setminus \{i\} \rightarrow y^{-1}(1))$ then i has to be excluded from $x^{-1}(1)$. Finally, the constraint $(\sum_{k \in \mathcal{I}} x_k = 1)$ only ensures that $x^{-1}(1)$ is not empty, because an empty-set is always generator.

Because of the closeness and generator constraints, the size of the formula to handle the MNRs became even larger. For the dataset `Retail` we move from a formula of size 63Gb for mining association rules, to a formula of size 187Gb for mining MNRs.

3.7 Conclusion

In this chapter we have presented some existing declarative approaches for itemset mining. We have presented the constraint programming and the SAT based approaches for itemset mining. We have introduced the basic constraint programming model for itemset mining and we have shown its limits. We have presented existing global constraints in itemset mining, namely, CLOSEDPATTERN and COVERSIZE. We also have presented some of the SAT models for itemset mining and we have shown that these models suffer from a memory issue. Other than itemset mining, declarative approaches are used in data mining including sequence mining, graph mining, mining from uncertain datasets, clustering, etc.

Recently, declarative approaches have shown interesting results in data mining. Hence, developing declarative approaches to solve some data mining problems seems to be a promising field. In the next part of this manuscript, we introduce our contributions in declarative itemset mining.

Part II

Contributions

IV

Chapter 4: Constraint Programming for Association Rules

4.1 Introduction

Mining association rules is one of the most studied problems in data mining. It aims at discovering interesting regularities between items in large-scale datasets. An Association Rule (AR) captures an information of the kind *"if we have A and B, the chances to have C are high"*. Nowadays, a broad spectrum of application domains ask for this kind of information on a variety of datasets.

In this chapter, we propose a full CP model for finding ARs. We use global constraints to ensure frequency and confidence of the rules. We take the frequency global constraint from existing literature (i.e., CoverSize constraint (Schaus et al. [2017])). For the confidence we need to introduce a new global constraint Confident for ensuring the minimum confidence of the extracted rules. We show that domain consistency on Confident is NP-hard. We then propose a non-complete propagator and a decomposition for Confident. We show that our CP model can easily be extended for taking into account any kind of user's constraints, such as cardinality of the rule, mandatory or forbidden items in the rule, etc. We show that our CP model is also able to capture the notion of MNR. For this, we introduce a new global constraint Generator for mining generators. We provide a polynomial algorithm achieving domain consistency on Generator. We then give a discussion on constrained MNRs. Experiments on several known large-scale datasets show the effectiveness of our global constraints and CP model.

This chapter is organized as follows. Section 4.2 presents our CP model for computing association rules. Section 4.3 defines the global constraint Confident, a global constraint for ensuring the confidence of a rule. Section 4.4 presents the extension of our CP model to compute MNRs. The global constraint Generator, which is needed in that model, is

defined and a propagator is proposed in Section 4.5. Section 4.6 shows how to extend our CP model to compute association rules with various kinds of user's constraints. We give discussion on constrained MNRs in Section 4.7. Finally, before concluding this chapter, Section 4.8 reports experiments.

4.2 A CP Model for Association Rules

In this section we present CP-RULE, a CP model for mining association rules. We introduce three vectors x, y and z of n Boolean variables, where x_i, y_i and z_i respectively represent the presence of item i in the body of the rule, in the head of the rule, and in the rule as a whole.

The model CP-RULE should be specified so that for any assignment on x, y, z that is a solution, $x^{-1}(1) \rightarrow y^{-1}(1)$ is a valid association rule.

CP-RULE involves five types of constraints:

$$\text{CP-RULE}_{\mathcal{D},s,c}(x,y,z) = \begin{cases} \forall i \in \mathcal{I} : \ \neg x_i \vee \neg y_i & (1) \\ \bigvee_{i \in \mathcal{I}} \ y_i & (2) \\ \forall i \in \mathcal{I} : \ z_i \iff x_i \vee y_i & (3) \\ \text{CONFIDENT}_{\mathcal{D},c}(x,y) & (4) \\ \text{FREQUENT}_{\mathcal{D},s}(z) & (5) \end{cases}$$

The role of each type of constraint is the following:

(1) ensures that a given item cannot be both in the body and in the head of a rule;
(2) ensures that the head of a rule is not empty;
(3) is a channelling constraint ensuring that $z^{-1}(1) = x^{-1}(1) \cup y^{-1}(1)$;
(4) is a global constraint that ensures that the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ is confident w.r.t. c;
(5) is a global constraint that ensures that $z^{-1}(1)$ is frequent w.r.t. s.

The constraint $\text{FREQUENT}_{\mathcal{D},s}$ has already been studied in the literature on itemset mining (Raedt et al. [2008], Schaus et al. [2017]). The constraint $\text{CONFIDENT}_{\mathcal{D},c}$, however, does not exist yet. We define it in the next section.

4.3 The Global Constraint CONFIDENT

In this section, we present a new global constraint for ensuring the confidence of an association rule. We prove that it is unfortunately NP-hard to enforce domain consistency on this constraint. As a result, we propose a non-complete filtering algorithm for CONFIDENT, and a decomposition using existing constraints, which is semantically equivalent to the CONFIDENT constraint.

Definition 4.1 (CONFIDENT constraint)**.** Let x and y be two vectors of Boolean variables. Let $\mathcal{D}$ be a dataset and c a minimum confidence threshold. The constraint $\text{CONFIDENT}_{\mathcal{D},c}(x,y)$ holds if and only if $conf(x^{-1}(1) \rightarrow y^{-1}(1)) \geq c$.

Example 4.1. Consider the dataset in Table 1.1 with $c = 80\%$.
CONFIDENT$_{\mathcal{D},80\%}([0,0,0,1,0],[1,1,0,0,0])$ is satisfied because the rule $D \rightarrow AB$ has a confidence of $100\% \geq c$. CONFIDENT$_{\mathcal{D},80\%}([1,0,0,0,1],[0,1,0,1,0])$ is not satisfied because the rule $AE \rightarrow BD$ has a confidence of $66.66\% < c$.

We now prove that enforcing domain consistency on the constraint CONFIDENT in NP-hard. This rules out the hope for a complete propagator for CONFIDENT.

Theorem 4.1. Enforcing domain consistency on the constraint CONFIDENT$_{\mathcal{D},c}$ is NP-hard.

Proof. From (Bessiere et al. [2007]) we know that if it is NP-complete to decide whether there exists a valid assignment satisfying a global constraint, then, enforcing domain consistency on this constraint is NP-hard. We then prove that it is NP-complete to decide if there exists a valid assignment for CONFIDENT$_{\mathcal{D},c}$.
Membership. Given the constraint CONFIDENT$_{\mathcal{D},c}(x,y)$ and a valid assignment on x and y, we traverse the table $\mathcal{D}$ and compute the size of the covers of the itemsets $x^{-1}(1)$ and $x^{-1}(1) \cup y^{-1}(1)$. This is linear in $|\mathcal{D}|$. We then compute the ratio $\frac{|cover(x^{-1}(1) \cup y^{-1}(1))|}{|cover(x^{-1}(1))|}$ and compare it to c to decide if the assignment satisfies the constraint. This is linear in $log|\mathcal{D}|$.
Completeness. We reduce 3SAT to the problem of deciding if there exists a valid assignment satisfying CONFIDENT$_{\mathcal{D},c}(x,y)$. Given a 3SAT formula F on the set $V = \{v_1, \ldots, v_n\}$ of Boolean variables, we construct the following instance of the constraint CONFIDENT$_{\mathcal{D},c}$. For clarity purpose, we denote the items in the transaction table $\mathcal{D}$ by `pos1`, `neg1`, $\cdots$, `posn`, `negn`, `z`. Thus, $x = \{x_{pos1}, x_{neg1}, \ldots, x_{posn}, x_{negn}, x_z\}$ and $y = \{y_{pos1}, y_{neg1}, \ldots, y_{posn}, y_{negn}, y_z\}$. Domains are defined by $dom(x_{posi}) = \{0,1\}, \forall i$, $dom(x_{negi}) = \{0,1\}, \forall i$, $dom(x_z) = \{0\}$, $dom(y_{posi}) = \{0\}, \forall i$, $dom(y_{negi}) = \{0\}, \forall i$ and $dom(y_z) = \{1\}$. We denote by `All` the set of all items. The confidence ratio c is set to 0.5.

The transactions table $\mathcal{D}$ is:

1. `All` $\setminus \{$`z`$\}$ (n times)
2. `All` $\setminus \{$`posi`$\}, \forall v_i \in V$
3. `All` $\setminus \{$`negi`$\}, \forall v_i \in V$
4. `All` $\setminus \{$`posi`, `negi`, `z`$\}, \forall v_i \in V$ (2 times)
5. `All` $\setminus \{it_1, it_2, it_3,$ `z`$\}$, for each clause cl in F, where $it_i =$ `posj` if the ith literal in cl is v_j, $it_i =$ `negj` if the ith literal in cl is $\neg v_j$.

For instance, if $F = \{l_1 \vee \neg l_2 \vee l_3\}$ with $n = 3$, $\mathcal{D}$ is:

```
t1  pos1 |neg1 |pos2 |neg2 |pos3 |neg3 |    |
t2  pos1 |neg1 |pos2 |neg2 |pos3 |neg3 |    |
t3  pos1 |neg1 |pos2 |neg2 |pos3 |neg3 |    |
t4       |neg1 |pos2 |neg2 |pos3 |neg3 |  z |
t5  pos1 |     |pos2 |neg2 |pos3 |neg3 |  z |
t6       |     |pos2 |neg2 |pos3 |neg3 |    |
t7       |     |pos2 |neg2 |pos3 |neg3 |    |
```

```
t8  pos1 |neg1 |      |neg2 |pos3 |neg3 | z  |
t9  pos1 |neg1 |pos2  |      |pos3 |neg3 | z  |
t10 pos1 |neg1 |      |      |pos3 |neg3 |    |
t11 pos1 |neg1 |      |      |pos3 |neg3 |    |
t12 pos1 |neg1 |pos2  |neg2  |     |neg3 | z  |
t13 pos1 |neg1 |pos2  |neg2  |pos3 |     | z  |
t14 pos1 |neg1 |pos2  |neg2  |     |     |    |
t15 pos1 |neg1 |pos2  |neg2  |     |     |    |
t16      |neg1 |pos2  |      |     |neg3 |    |
```

Suppose a formula F is satisfiable. Let us denote by S a solution of F. We construct the valid assignment on x, y such that $x_{posi} = 1$ and $x_{negi} = 0$ for each i such that $S[v_i] = 1$, and $x_{posi} = 0$ and $x_{negi} = 1$ for each i such that $S[v_i] = 0$. Bear in mind that x_z and y are already assigned as they have a singleton domain. By construction of $\mathcal{D}$, $x^{-1}(1) \cup y^{-1}(1)$ appears in n transactions (2) and (3). By construction again, $x^{-1}(1)$ appears in the n transactions where $x^{-1}(1) \cup y^{-1}(1)$ appears plus the n transactions (1). $x^{-1}(1)$ does not appear in any transaction (4) because they all miss `posi` and `negi` for some i, whereas $x^{-1}(1)$ contains `posi` or `negi` for all i. Finally, as S satisfies F, $x^{-1}(1)$ does not appear in any transaction (5) because these transactions all miss at least the item of $x^{-1}(1)$ corresponding to the literal satisfying the clause. As a result, the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ has confidence $\frac{n}{2n} = 0.5$, and our assignment on (x, y) satisfies the constraint $\text{CONFIDENT}_{\mathcal{D},c}(x, y)$.

Suppose now that A is a valid assignment on x, y satisfying the constraint $\text{CONFIDENT}_{\mathcal{D},c}(x, y)$. Remember that $y^{-1}(1)$ necessarily contains `z`, so $x^{-1}(1)$ does not. Hence, $x^{-1}(1)$ appears at least in the n transactions (1) where $y^{-1}(1)$ does not appear. Now, $y^{-1}(1)$ only appears in transactions (2) and (3) because it contains `z`. Thus, $x^{-1}(1)$ must appear in at least n transactions (2) and (3) to reach the confidence of 50%. For a given i, $x^{-1}(1)$ must contain at least one among `posi` and `negi`, otherwise the two corresponding transactions (4) would cover $x^{-1}(1)$ and not $y^{-1}(1)$, making confidence impossible to reach. Thus, $x^{-1}(1)$ can (and must) appear in exactly n transactions (2) and (3), which means that for each i, exactly one among `posi` and `negi` is in $x^{-1}(1)$. We then can build the mapping from the assignment A on x, y to the instantiation S on $v_1, \ldots, v_n$ such that $S[v_i] = 1$ if $A[x_{posi}] = 1$, and $S[v_i] = 0$ if $A[x_{negi}] = 1$. We have n transactions (1-4) covering $x^{-1}(1) \cup y^{-1}(1)$ and $2n$ covering $x^{-1}(1)$. As transactions (5) do not contain `z`, they must not cover $x^{-1}(1)$, otherwise confidence cannot be reached. As a result, for every transaction (5), $x^{-1}(1)$ necessarily contains at least one item (other than `z`) which is not in the transaction. By construction of transactions (5) and thanks to the mapping from A to S, this item corresponds to the truth value of a Boolean variable that satisfies the clause of F associated with the transaction. Therefore, F is satisfiable.

Consequently, deciding if there exists a valid assignment satisfying the constraint $\text{CONFIDENT}_{\mathcal{D},c}$ is NP-complete, and domain consistency on $\text{CONFIDENT}_{\mathcal{D},c}$ is NP-hard. □

Theorem 4.1 tells us that we cannot efficiently enforce domain consistency on $\text{CONFIDENT}_{\mathcal{D},c}$ unless $P = NP$. We thus propose weaker propagators for $\text{CONFIDENT}_{\mathcal{D},c}$.

A propagation rule. We first propose a filtering rule for $\text{CONFIDENT}_{\mathcal{D},c}$. The rule is activated only if x is entirely instantiated.

Proposition 4.1. Given a complete assignment on x and a partial assignment on y, for any $i \in y^{-1}(*)$, if $conf(x^{-1}(1) \rightarrow y^{-1}(1) \cup \{i\}) < c$ then we can remove value 1 from $dom(y_i)$.

Proof. We know that $conf(x^{-1}(1) \rightarrow y^{-1}(1)) = \frac{|cover(x^{-1}(1) \cup y^{-1}(1))|}{|cover(x^{-1}(1))|} = \frac{1}{|cover(x^{-1}(1))|} \times |cover(x^{-1}(1) \cup y^{-1}(1))|$. As x is entirely instantiated, $\frac{1}{|cover(x^{-1}(1))|}$ is constant (let us call it k). Then $conf(x^{-1}(1) \rightarrow y^{-1}(1)) = k \times |cover(x^{-1}(1) \cup y^{-1}(1))| = k \times freq(x^{-1}(1) \cup y^{-1}(1))$. We want to mine confident rules, i.e. $k \times freq(x^{-1}(1) \cup y^{-1}(1)) \geq c$. This is $freq(x^{-1}(1) \cup y^{-1}(1)) \geq c/k$. Using the anti-monotone property of the frequency, if the itemset $x^{-1}(1) \cup y^{-1}(1) \cup \{i\}$ ($y_i = 1$) is infrequent w.r.t. c/k then the item i is discarded ($1 \notin dom(y_i)$). □

Example 4.2. Consider the dataset in Table 1.1 with $c = 70\%$. Given the complete assignment on x, $x^{-1}(1) = \{A\}$, $x^{-1}(0) = \{B, C, D, E\}$ and $x^{-1}(*) = \emptyset$, and the partial assignment on y, $y^{-1}(1) = \{B\}$, $y^{-1}(0) = \{A\}$ and $y^{-1}(*) = \{C, D, E\}$, the filtering rule in Proposition 4.1 filters 1 from $dom(y_C)$ because the rule $A \rightarrow BC$ is not confident w.r.t. c ($conf(A \rightarrow BC) = 50\% < c$).

A decomposition. We also propose an incomplete propagation operated by a decomposition of $\text{CONFIDENT}_{\mathcal{D},c}$ using the global constraint COVERSIZE (Schaus et al. [2017]), which is able to capture the frequencies of x and $z = x \cup y$. The decomposition of $\text{CONFIDENT}_{\mathcal{D},c}$ is as follows.

$$\text{CONFIDENT}_{\mathcal{D},c}(x, y) \equiv \begin{cases} \text{COVERSIZE}_{\mathcal{D}}(x, p) \\ \text{COVERSIZE}_{\mathcal{D}}(x \cup y, q) \\ \frac{q}{p} \geq c \end{cases}$$

In this decomposition, p represents the frequency of $x^{-1}(1)$ and q represents the frequency of $z^{-1}(1)$. The fraction $\frac{q}{p}$ represents the confidence of the rule $x^{-1}(1) \rightarrow y^{-1}(1)$. We simply ensure that the fraction $\frac{q}{p}$ is greater or equals the confidence threshold c using an inequality constraint.

4.4 A CP model for computing MNRs

In this section we show how our CP-RULE model can be extended to return only MNRs (see Definition 1.10). For this we use a global constraint that we define later.

According to Proposition 1.4, our CP-RULE model can be extended to a model able to extract MNRs by adding two constraints:

$$\text{MNRULE}_{\mathcal{D},s,c}(x, y, z) = \begin{cases} \text{CP-RULE}_{\mathcal{D},s,c}(x, y, z) & (1) \\ \text{GENERATOR}_{\mathcal{D}}(x) & (2) \\ \text{CLOSED}_{\mathcal{D}}(z) & (3) \end{cases}$$

where:

(1) CP-RULE ensures that the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ is valid;
(2) is a global constraint that ensures that the itemset $x^{-1}(1)$ is a generator;
(3) is a global constraint that ensures that the itemset $z^{-1}(1)$ is closed.

The constraint CLOSED$_{\mathcal{D}}$ has already been studied in the literature on itemset mining (Lazaar et al. [2016], Schaus et al. [2017]). The constraint GENERATOR$_{\mathcal{D}}$, however, does not exist yet. We define it in the next section.

4.5 The Global Constraint GENERATOR

The new global constraint GENERATOR is introduced in this section. GENERATOR is used to mine itemsets that are generator. We propose a complete polynomial algorithm achieving DC on GENERATOR.

Definition 4.2 (GENERATOR constraint)**.** Let x be a vector of Boolean variables and $\mathcal{D}$ be a dataset. The global constraint GENERATOR$_{\mathcal{D}}(x)$ holds if and only if $x^{-1}(1)$ is a generator.

Example 4.3. Consider the dataset in Table 1.1. GENERATOR$_{\mathcal{D}}([1,1,0,0,0])$ is satisfied because AB does not have any subset with the same frequency. GENERATOR$_{\mathcal{D}}([0,1,0,0,1])$ is not satisfied because BE has a subset with the same frequency ($freq(BE) = freq(B)$).

The propagator we propose for the GENERATOR constraint is based on the property in Proposition 1.2.

Algorithm. The propagator for the global constraint GENERATOR is presented in Algorithm 1. Algorithm 1 takes as input the variables x. Algorithm 1 starts by computing the cover of the itemset $x^{-1}(1)$ and stores it in `cover` (line 3). Then, for each item $j \in x^{-1}(1)$, Algorithm 1 computes the cover of the subset $x^{-1}(1) \setminus \{j\}$, and stores it in `cov[j]` (line 5). Algorithm 1 checks if a subset has the same frequency as $x^{-1}(1)$, if so the constraint is violated and a fail is returned (line 6). Algorithm 1 can then remove items i that cannot belong to a generator containing $x^{-1}(1)$. To do that, we start by comparing the cover of $x^{-1}(1) \cup \{i\}$ (i.e., `cover` $\cap\, cover(i)$) with the cover of $x^{-1}(1)$ (i.e., `cover`) (line 8). If they have equal size (i.e., same frequency), we remove i from the possible items, that is, we remove 1 from $dom(x_i)$ and continue to the next item (line 9). Then, for every item j in $x^{-1}(1) \cup \{i\}$, we compare the cover of $x^{-1}(1) \cup \{i\}$ (i.e., `cover` $\cap\, cover(i)$) to the cover of $x^{-1}(1) \cup \{i\} \setminus \{j\}$ (i.e., `cov[j]` $\cap\, cover(i)$) (line 11). If they have equal size (i.e., same frequency), we remove i from the possible items, that is, we remove 1 from $dom(x_i)$ and break the loop (line 12).

Example 4.4. Consider the dataset in Table 3.1. Given the partial assignment $x^{-1}(1) = \{B\}$, $x^{-1}(0) = \{D\}$ and $x^{-1}(*) = \{A, C, E\}$, the Algorithm 1 filters 1 from $dom(x_E)$ because $freq(BE) = freq(B)$.

Theorem 4.2. The propagator in Algorithm 1 enforces domain consistency on the constraint GENERATOR.

Algorithm 1: Propagator for GENERATOR

```
1  InOut: x = {x_1 ... x_n}: Boolean item variables;
2  begin
3      cover ← cover(x^{-1}(1));
4      foreach j ∈ x^{-1}(1) do
5          cov[j] ← cover(x^{-1}(1) \ {j});
6          if |cover| = |cov[j]| then return failure ;
7      foreach i ∈ x^{-1}(*) do
8          if |cover ∩ cover(i)| = |cover| then
9              dom(x_i) ← dom(x_i) \ {1}; continue;
10         foreach j ∈ x^{-1}(1) do
11             if |cover ∩ cover(i)| = |cov[j] ∩ cover(i)| then
12                 dom(x_i) ← dom(x_i) \ {1}; break;
```

Proof. We first prove that the value 0 for a variable x_i such that $i \in x^{-1}(*)$ always belongs to a solution of the constraint GENERATOR, and so cannot be pruned by domain consistency. Suppose $i \in x^{-1}(*)$. If $x^{-1}(1)$ is a generator, removing the value 0 from $dom(x_i)$ increases $x^{-1}(1)$ to $x^{-1}(1) \cup \{i\}$, and then $x^{-1}(1)$ cannot be returned as a generator, which contradicts the hypothesis. Suppose now that $x^{-1}(1)$ is not a generator. We know from Proposition 1.2 that for any $Q \supsetneq x^{-1}(1)$, Q is not a generator. Thus, $x^{-1}(1) \cup \{i\}$ cannot belong to any generator, and value 0 cannot be pruned from $dom(x_i)$.

We now prove that Algorithm 1 prunes value 1 from $dom(x_i)$ exactly when i cannot belong to a generator containing $x^{-1}(1)$. Suppose value 1 of x_i is pruned by Algorithm 1. This means that one of the tests in line 8 or 11 was true, that is, there exists a sub-itemset of $x^{-1}(1) \cup \{i\}$ with the same frequency as $x^{-1}(1) \cup \{i\}$. Thus, by definition, $x^{-1}(1) \cup \{i\}$ does not belong to any generator. Suppose now that value 1 of x_i is not pruned. From the lines 8 and 11, we deduce that there does not exist any subset of $x^{-1}(1) \cup \{i\}$ with the same frequency as $x^{-1}(1) \cup \{i\}$. Thus $x^{-1}(1) \cup \{i\}$ is a generator and value 1 of x_i is domain consistent.

□

Theorem 4.3. Given a transaction dataset $\mathcal{D}$ of n items and m transactions, Algorithm 1 has an $O(n^2 \times m)$ time complexity.

Proof. Computing the size of the cover of an itemset is in $O(n \times m)$. Line 5 is called at most n times, leading to a time complexity of $O(n^2 \times m)$. The test at line 8 is done at most n times. The cover of $x^{-1}(1) \cup \{i\}$ is computed in $O(m)$ thanks to `cover`. The test at line 11 is done at most n^2 times. The covers of $x^{-1}(1) \cup \{i\}$ and $x^{-1}(1) \cup \{i\} \setminus \{j\}$ at line 11 are computed in $O(m)$ thanks to the `cover` and `cov` data structures. Thus, the time complexity of lines 7-12 is bounded above by $O(n^2 \times m)$. As a result, Algorithm 1 has an $O(n^2 \times m)$ time complexity. □

Note that without the use of the cov structure (that is, by recomputing $cover(x^{-1}(1) \setminus \{j\})$ at each execution of the loop at line 7), the time complexity becomes $O(n^3 \times m)$. However, this version is less memory consuming and can be more efficient in practice. This is especially true when the subset that has the same frequency as $x^{-1}(1) \cup \{i\}$ is $x^{-1}(1)$ (i.e., when the test at line 8 is true), which reduces the time complexity to $O(n \times m)$.

It is also important to stress that domain consistency on GENERATOR does not depend on $x^{-1}(0)$. Thus, Algorithm 1 is not called during the solving process when a variable is instantiated to zero.

4.6 On Constrained Association Rules

In many practical cases, the user asks for association rules satisfying extra constraints in addition to frequency and confidence. Such user's constraints are handled by data mining researchers, whenever possible, with (i) a pre-processing step reducing the dataset; (ii) a filtering integrated in the specialized algorithm; (iii) a post-processing step to filter out the undesirable rules. In this section, we show how flexibly the CP-RULE model can be augmented to express the kind of association rules the user is interested in. We illustrate this flexibility on a few examples of constraints that are common properties of rules a user may be interested in (Wojciechowski and Zakrzewicz [2002]).

4.6.1 Mandatory / forbidden items in a rule

The user may ask for rules that involve a particular set $\mathcal{M}$ of mandatory items in the body (resp. in the head, or in the rule as a whole). With the CP approach, such constraint can simply be added to the CP-RULE model as follows:

$$mand_{\mathcal{M}}(x) \equiv (\forall i \in \mathcal{M} : x_i = 1)$$

where x is respectively replaced by y or z if $\mathcal{M}$ must be in the head or in the rule as a whole.

Similarly, the user may ask for rules that do not involve a set of forbidden items ($\mathcal{F}$) in the body (resp. in the head, or in the rule as a whole). In CP-RULE, we can express this constraint as follows:

$$forb_{\mathcal{F}}(x) \equiv (\forall i \in \mathcal{F} : x_i = 0)$$

where x is respectively replaced by y or z if $\mathcal{F}$ must be forbidden from the head or from the rule as a whole.

For instance, the user can ask for rules with a head containing a set of items $\mathcal{M}$ and a body not containing a set of items $\mathcal{F}$. The two following constraints are added to the

CP-RULE model:

$$forb_{\mathcal{F}}(x) \wedge mand_{\mathcal{M}}(y)$$

.

4.6.2 Cardinality constraints

The user may also be interested in constraints on the cardinality of the body of the rule (and/or the head, and/or the rule as a whole):

— $atLeast$ constraint: A rule (body and/or head) contains at least lb items:

$$atLeast_{lb}(x) \equiv \sum_{i \in \mathcal{I}} x_i \geq lb$$

— $atMost$ constraint: A rule (body and/or head) contains at most ub items:

$$atMost_{ub}(x) \equiv \sum_{i \in \mathcal{I}} x_i \leq ub$$

— $Exactly$ constraint: A rule (body and/or head) contains exactly v items:

$$Exactly_{v}(x) \equiv \sum_{i \in \mathcal{I}} x_i = v$$

where x is respectively replaced by y or z if the cardinality constraint is on the head or on the rule as a whole.

For instance, when a user asks for ARs with a size of v, a body of at most ub and a head of at least lb, the CP-RULE model is extended with the following constraints:

$$Exactly_{v}(z) \wedge atMost_{ub}(x) \wedge atLeast_{lb}(y)$$

4.7 On Constrained MNRs

Constraints added to the model MNRULE can interfere with the maximality of $x^{-1}(1) \cup y^{-1}(1)$ (i.e., $x^{-1}(1) \cup y^{-1}(1)$ being closed) and the minimality of $x^{-1}(1)$ (i.e., $x^{-1}(1)$ being generator) (see Section 1.6.1).

At first glance, it seems that our CP model for mining MNRs cannot be extended with anti-monotone constraints on the body of the rule because $x^{-1}(1) \cup y^{-1}(1)$ is closed and the body is in $x^{-1}(1) \cup y^{-1}(1)$. But the body being generator guarantees the safety of our model in taking in consideration anti-monotone constraints on the body and monotone constraints on the head.

In this section we prove that anti-monotone constraints on the body and monotone constraints on the head are safe on MNRULE. That is, there always exist MNRs satisfying the constraints and dominating the satisfied rules (see Definition 1.9).

Theorem 4.4. Given a valid non MNR rule $r : X \rightarrow Y$ satisfying c_1 an anti-monotone constraint on X and c_2 a monotone constraint on Y, there always exists an MNR $r' : X' \rightarrow Y'$ satisfying $c_1(X')$ and $c_2(Y')$ such that $r' \succ r$.

Proof. Suppose that $r : X \rightarrow Y$ is not MNR satisfying $c_1(X)$ an anti-monotone constraint and $c_2(Y)$ a monotone constraint. We prove that there always exists an MNR r' dominating r and satisfying c_1 and c_2. From the definition of MNRs we know that if r is not MNR then there exists an MNR $r' : X' \rightarrow Y'$ such that $r' \succ r$, i.e. $freq(r') = freq(r)$ and $conf(r') = conf(r)$ and $X' \subseteq X$, $Y' \supseteq Y$. As $c_1(X)$ is a satisfied anti-monotone constraint and $X' \subseteq X$ then $c_1(X')$ is satisfied too. Similarly, as $c_2(Y)$ is a satisfied monotone constraint and $Y' \supseteq Y$ then $c_2(Y')$ is satisfied too. Hence, the rule r' is an MNR satisfying the constraints and dominating the rule r. □

4.8 Experimental Evaluation

We made several experiments to evaluate our constraints and our CP model for mining pure and constrained association rules. We compared to the state of the art approaches.

4.8.1 Benchmark Datasets

We selected several real-sized datasets from the FIMI repository. [1] These datasets have various characteristics representing different application domains. Table 4.1 reports for each dataset the number of transactions $|\mathcal{T}|$, the number of items $|\mathcal{I}|$, the average size of transactions $\overline{|\mathcal{T}|}$, the density ρ (i.e., $\overline{|\mathcal{T}|}/|\mathcal{I}|$), and its application domain. The datasets are presented by increasing size $|\mathcal{I}| \times |\mathcal{T}|$. We selected datasets of various size and density. Some datasets, such as `Zoo` and `Chess`, are very dense (resp. 44% and 49%). Others, such as `T10` and `Retail`, are very sparse (resp. 1% and 0.06%). The sizes of these datasets vary from around $4,000$ to more than 10^9.

4.8.2 Experimental Protocol

The implementation of our CP models and constraint propagators were carried out in the `Oscar` solver using Scala. [2] The code is publicly available. [3]

For each dataset, an instance is characterized by its minimum frequency threshold (i.e., s). For instance, `Zoo_5` denotes the instance of the `Zoo` dataset with a minimum frequency threshold of 5.

1. fimi.ua.ac.be/data/
2. bitbucket.org/oscarlib/oscar/
3. gite.lirmm.fr/belaid/cp4ar

Table 4.1 – *Dataset Characteristics.*

Dataset	$\|\mathcal{T}\|$	$\|\mathcal{I}\|$	$\overline{\|\mathcal{T}\|}$	$\rho(\%)$	Type of Data
Zoo	101	36	16	44	Animals data
Vote	435	48	16	33	Vote data
Anneal	812	93	42	45	Anneal data
Chess	3,196	75	37	49	Game steps
Mushroom	8,124	119	23	19	Species of mushrooms
Connect	67,557	129	43	33	Game steps
T10	100,000	1,000	10	1	Synthetic dataset
T40	100,000	1,000	40	4	Synthetic dataset
Pumsb	49,046	2,113	74	3	Census data
Retail	88,162	16,470	10	0.06	Retail market basket data

T10 = T10I4D100K T40 = T40I10D100K

All experiments were conducted on an Intel core i7, 2.2Ghz with a RAM of 8Gb and a timeout of one hour.

4.8.3 Mining Frequent Generators

To evaluate the new global constraint GENERATOR, we compare our CP approach for mining frequent generators with TALKY-G (Szathmary et al. [2009]). We used the implementation of TALKY-G publicly available in the CORON Data Mining Platform.[4] Our approach combines the constraint $\text{GENERATOR}_{\mathcal{D}}$ with $\text{COVERSIZE}_{\mathcal{D}}(x, p)$ and $p \geq s$ to ensure the frequency. After a few preliminary tests, we decided to use *smallest item frequency* first as variable ordering heuristic and *largest value* first as value ordering heuristic. We have selected for every dataset frequency thresholds to have different numbers of frequent generators. The execution time for every approach and the number of frequent generators, $\#FG$, are reported in Table 4.2.

The first observation to notice, and as expected, TALKY-G outperforms the CP approach for mining frequent generators. However, CP is very competitive and even better than TALKY-G on some instances (e.g., Zoo_1, T10_500 and Retail_45). This is specially true when the number of frequent generators is small (i.e., less than $20K$ solutions).

Note that we are using the less memory version of our propagator presented in Algorithm 1. This version is efficient in practice. On Anneal_244, for instance, all the filterings of Algorithm 1 were conducted because of $x^{-1}(1)$ having the same frequency as $x^{-1}(1) \cup \{i\}$ (the test in line 8 of Algorithm 1 is true).

4. coron.loria.fr

Table 4.2 – CP *vs* TALKY-G *for mining frequent generators (time in seconds)*

Instance	CP	TALKY-G	$\#FG$
Zoo_1	**0.44**	0.70	9,978
Vote_40	0.96	**0.31**	42,793
Vote_4	3.80	**0.88**	258,017
Chess_1918	2.90	**0.42**	98,419
Chess_959	152.78	**29.29**	5,601,829
Chess_640	968.60	**333.58**	25,031,186
Anneal_244	5.93	**1.38**	343,272
Anneal_41	35.93	**25.56**	3,510,475
Mushroom_82	2.73	**0.45**	69,966
Mushroom_9	8.43	**1.28**	234,231
Connect_47290	7.52	0.48	35,876
Connect_20268	114.39	**5.24**	460,357
T10_500	**1.46**	1.95	1,071
T10_10	324.64	**78.12**	307,467
T40_5000	**0.47**	0.53	317
T40_1000	30.74	**3.99**	65,237
Pumsb_39237	24.39	**1.36**	67,836
Pumsb_29428	1789.20	**72.32**	2,853,042
Retail_45	**22.39**	45.54	19,115
Retail_9	161.91	**114.22**	191,266

4.8.4 CONFIDENT: Dedicated Propagator vs Decomposition

For propagating the constraint CONFIDENT, we have implemented both propositions presented in Section 4.3. In this section we compare two CP models for mining ARs. The one that uses the filtering rule presented in Proposition 4.1 as propagator for CONFIDENT (i.e., F-RULE) and the one that uses a decomposition using COVERSIZE (i.e., DECOMPOSITION). We have selected 3 datasets, small, middle and huge with a fixed frequency threshold and we have variated the minimum confidence threshold, i.e. c. In Table 4.3 we report the execution time for both approaches. We also report the number of nodes to evaluate the level of filtering conducted by each approach.

Clearly, the use of the decomposition provides us with a better resolution time due the practical efficiency of the propagator of COVERSIZE. The level of filtering is also slightly better. The number of nodes being the same for Vote_22 with $c = 70\%$, DECOMPOSITION outperforms F-RULE with $1,276$ nodes for Vote_22 with $c = 100\%$. In fact, DECOMPOSITION can filter 0 from all $dom(x_i)$ if that is the only way to ensure that $q \geq p \times c$. This filtering occurs rarely but it explains the slight difference in the number of nodes.

Hence, for the next experiments we use the decomposition to replace the constraint CONFIDENT in our model.

Table 4.3 – DECOMPOSITION *vs* F-RULE *(time in seconds)*

Instances	$c(\%)$	DECOMPOSITION		FILTER	
		$Time$	$\#Node$	$Time$	$\#Node$
Vote_22	70	**28.31**	**20,113,200**	35.68	**20,113,200**
	90	**6.43**	**4,167,824**	9.48	4,167,968
	100	**2.18**	**709,352**	2.32	710,628
Mushroom_1500	70	**8.02**	**2,934,558**	16.28	2,934,628
	90	**6.11**	**2,054,332**	11.15	2,054,428
	100	**4.23**	**1,362,110**	8.32	1,362,318
T40_1300	70	**16.40**	**323,382**	39.77	**323,382**
	90	**13.16**	**218,690**	31.83	218,734
	100	**8.21**	**18,416**	21.06	20,062

4.8.5 Heuristic of Variable Selection for Mining Association Rules

In this section we compare the resolution time of our approach using different static variable ordering heuristics. We have tested all the combinations $\langle x, y, z\rangle$ to decide the order of vectors. The variables of a given vector are ordered in a *lexicographic* order and we use *smallest value* first as value ordering heuristic.

Table 4.4 presents the time, in seconds, for every variable ordering heuristic on every instance. For every dataset we have selected s to have a hard instance and we fixed c to 90%.

The first observation that we can draw from Table 4.4 is that the heuristics starting with the same vector perform almost the same (e.g., $\langle x, y, z\rangle$ and $\langle x, z, y\rangle$). It is clear that the heuristics starting with the body of the rule x outperform the others (the ones starting with y or z). If we compare the heuristics starting with x we notice that $\langle x, y, z\rangle$ is slightly better than $\langle x, z, y\rangle$ although $\langle x, z, y\rangle$ being competitive and even better on some instances (e.g., Anneal_650). As a result, we use the variable ordering heuristic $\langle x, y, z\rangle$ in the rest of our experiments.

Table 4.4 – *Variable ordering heuristics (time in seconds)*

Instances	$\langle x, y, z\rangle$	$\langle x, z, y\rangle$	$\langle z, x, y\rangle$	$\langle z, y, x\rangle$	$\langle y, x, z\rangle$	$\langle y, z, x\rangle$
Zoo_6	**43.45**	47.86	110.78	116.46	122.23	120.24
Vote_22	**6.43**	7.74	20.08	19.74	50.25	48.66
Anneal_650	208.87	**193.96**	224.82	275.86	310.43	344.83
Chess_1918	**68.74**	71.10	112.33	117.18	153.80	207.20
Mushroom_813	**50.67**	54.90	108.02	112.21	147.13	146.76
Connect_90	**125.48**	127.90	146.39	143.35	269.56	255.72
T10_20	63.35	**62.52**	82.08	82.72	340.66	278.65
T40_1000	134.64	**134.23**	173.43	167.53	410.14	386.49
Pumsb_39237	1567.72	**1552.79**	1627.22	1655.04	2522.13	2354.25
Retail_89	58.76	57.69	51.14	**50.85**	108.68	103.93

4.8.6 Mining Association Rules

In this experiment we compare our CP approach to the ECLAT-Z specialized algorithm for extracting ARs (Szathmary et al. [2007c]), and to a SAT-based approach (Boudane et al. [2016]) (note that this approach is hybrid and it uses constraint propagation to handle the cardinality constraints). We used the implementation of ECLAT-Z publicly available in the CORON Data Mining Platform. [5] In all the instances presented in the following experiments, the minimum confidence has been fixed to 90% so that we focus on highly confident rules. For each dataset, an instance is characterized by its minimum frequency threshold. For instance, `Zoo_51` denotes the instance of the `Zoo` dataset with a minimum frequency threshold of 51 (and always a minimum confidence of 90%). For each dataset, we have selected three instances according to the number of solutions. The first instance has less than $1,000$ ARs, the second instance has between $100,000$ and one million ARs, and the third instance has more than one million ARs. The only exception is the very large and sparse dataset `Retail`, for which we could not reach large number of solutions. Table 4.5 reports the CPU time, in seconds, for each approach on each selected instance. We also report the total number of ARs (#TOT) for each instance. [6]

The first observation that we can draw from Table 4.5 is that, as expected, the specialized algorithm ECLAT-Z performs very well. However, the CP approach is very competitive too, and sometimes faster than ECLAT-Z (see the first instance of each dataset, where only a few valid ARs exist). The explanation for this good behavior of CP is the strength of constraint propagation to rule out inconsistent parts of the search space. On the contrary, on an instance as loose as `Pumsb_39237`, with $49,000$ transactions, $2,113$ items, and around 20 million solutions, the CP solver is almost reduced to an enumerating process.

Concerning the SAT approach, on small and middle-sized datasets (i.e., from `Zoo` to `Connect`), we observe that SAT performs reasonably well, although being slower than CP on almost all the instances. On larger and larger datasets, the results of SAT become worse and worse. On very large datasets, SAT reaches the one hour timeout for `T10` and `T40` instances and out-of-memory on all the instances of `Pumsb` and `Retail`. The out-of-memory state is due to the fact that SAT generates huge propositional formulas to represent the dataset and the constraints (see Section 3.6.2). On the instances where we observe an out-of-memory, SAT generates formulas of size ranging from 14Gb to 63Gb.

Finally, it is important to note that even when CP reaches the timeout, it returns solutions on the fly before the timeout. This is because CP, as opposed to ECLAT-Z, does not need to build any complex data structure before starting the search for ARs. On `T40_100`, CP returns the first solution in only 1.25 seconds and returns more than 390 million solutions before the timeout, whereas ECLAT-Z does not return any AR to the user before having computed its whole structure, that is, no solution before the timeout on `T40_100`.

Our next experiment evaluates the behaviour of ECLAT-Z, CP and SAT when varying the minimum confidence (c). Figure 4.1 illustrates the time, in seconds, for each approach on each selected minimum confidence threshold on the instance `Vote_22`. From Figure 4.1

5. coron.loria.fr
6. This number is computed by releasing the timeout.

Table 4.5 – ECLAT-Z *vs* SAT *vs* CP *for extracting ARs (time in seconds)*

Instances	ECLAT-Z	SAT	CP	#TOT
Zoo_51	0.07	**0.02**	0.04	292
Zoo_31	**0.40**	0.92	0.82	198,971
Zoo_5	**24.99**	106.52	59.09	30,792,317
Vote_153	**0.04**	0.32	0.05	244
Vote_44	**1.81**	255.52	1.79	419,204
Vote_22	**5.66**	1054.81	6.43	2,075,212
Anneal_780	0.08	0.19	**0.06**	798
Anneal_731	**0.28**	1.11	1.17	174,710
Anneal_650	**93.96**	467.83	208.87	84,589,753
Chess_3037	0.08	0.56	**0.07**	474
Chess_2557	**0.62**	5.97	2.16	349,298
Chess_1918	**15.98**	260.21	68.74	17,522,446
Mushroom_4062	**0.10**	3.78	**0.10**	469
Mushroom_1747	**0.59**	51.87	1.43	112,826
Mushroom_813	**27.81**	686.39	50.67	14,331,056
Connect_66882	0.64	26.90	**0.04**	70
Connect_63504	**0.99**	95.97	7.70	201,928
Connect_60802	**3.03**	1126.91	125.48	3,640,704
T10_500	**0.29**	TO	2.71	532
T10_100	**1.63**	TO	9.92	160,171
T10_20	**15.12**	TO	63.35	1,303,932
T40_1600	**1.35**	TO	5.36	648
T40_1300	**1.73**	TO	13.16	101,588
T40_100	TO	TO	TO	$> 7.10^9$
Pumsb_47085	1.31	OOM	**0.11**	338
Pumsb_43651	**2.07**	OOM	8.39	135,677
Pumsb_39237	**34.35**	OOM	1567.72	19,749,382
Retail_441	1.59	OOM	**0.52**	37
Retail_265	**1.74**	OOM	2.27	75
Retail_89	**2.53**	OOM	58.76	255

TO**: timeout** OOM**: out-of-memory**

we can observe that our CP approach is the one that improves the most for mining highly confident rules. From the instance where $c = 10\%$ to the instance where $c = 100\%$, the CPU time for CP is reduced by 98% (from 189.56 to 2.46 seconds), whereas for ECLAT-Z it is reduced by 83% (from 28.75 to 4.61 seconds) and for SAT by only 36% (from 1578.35 to 1006.35 seconds). As a result, our CP approach is always better than SAT and becomes even better than ECLAT-Z with $c = 100\%$. This is due to the constraint propagation conducted by the constraints in the decomposition of CONFIDENT which provides CONFIDENT with an efficient filtering.

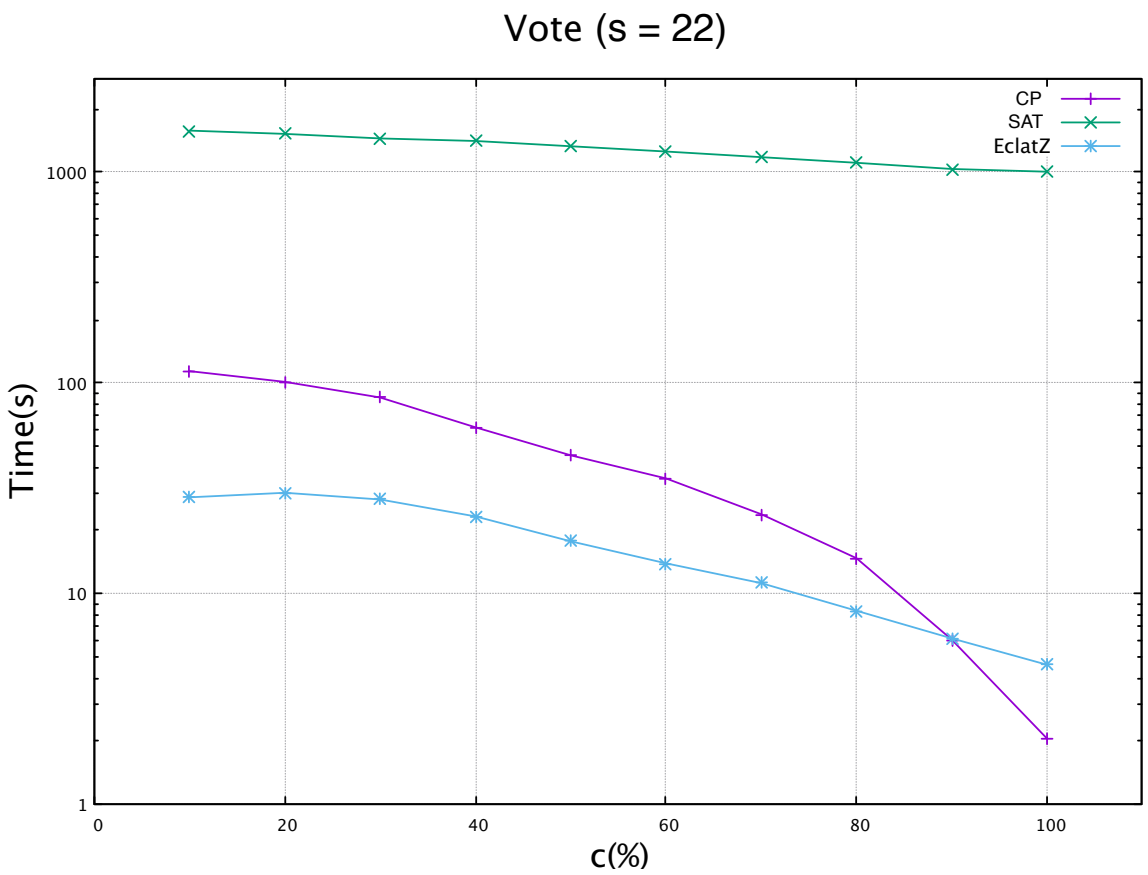


Figure 4.1 – `Vote_22`*:* ECLAT-Z *vs* SAT *vs* CP

4.8.7 Mining MNRs.

This experiment compares ECLAT-Z (Szathmary et al. [2007c]) to SAT (Boudane et al. [2017]) and CP for extracting MNRs. Table 4.6 reports the CPU time, in seconds, for each approach on each selected instance. We also report the total number of MNRs (#TOT).

For the MNRULE model presented in Section 4.4, we have used the global constraint GENERATOR presented in Section 4.5, and to ensure closeness, we have used the constraint COVERCLOSURE introduced in (Schaus et al. [2017]).

SAT wins only on the smallest dataset (i.e., `Zoo` instances). For larger datasets, the encoding of the MNR constraints in SAT can take three times more space than the encoding of the original AR problem and then can lead to an out-of-memory state. For instance, SAT encodes only the first constraints of the AR problem on a `Retail` instance with a formula of 63Gb (see Section 3.6.2). When moving to the MNR problem, we reach 187Gb.

ECLAT-Z can take significantly more time to enumerate MNRs than ARs. On the `T10_20` instance, it is easier for ECLAT-Z to enumerate more than 1 million ARs (15.12 seconds) than to enumerate 257 thousand MNRs (1131.39 seconds). The opposite is observed on CP. Such a performance is mainly due to the constraints GENERATOR and COVERCLOSURE that provide the CP solving process with more propagation to remove more inconsistent values in the search space. This is especially true when the number of non-

solutions to prune is important. Nevertheless, the Retail instances contradict this trend. This is explained by the fact that on Retail almost all ARs are MNRs whereas in average on all our instances only 7% of the ARs are MNRs. Hence, on Retail the constraints GENERATOR and COVERCLOSURE are redundant to the model for ARs (CP-RULE). Thus, the propagators of GENERATOR and COVERCLOSURE do not participate to the reduction of the search space. They just waste time. For instance, CP explores exactly the same search space ($2,750$ nodes) on Retail_265 to extract ARs and MNRs (because all ARs are MNRs).

Again, on the timeout instance T40_100, CP returns its first solution in 1.23 seconds and returns more than 160 million MNRs before the timeout. ECLAT-Z is not able to return any MNR before the timeout.

Table 4.6 – ECLAT-Z *vs* SAT *vs* CP *for extracting MNRs (time in seconds)*

Instances	ECLAT-Z	SAT	CP	#TOT
Zoo_51	0.04	**0.03**	0.06	177
Zoo_31	0.39	**0.08**	0.17	2,261
Zoo_5	8.21	**0.47**	0.72	13,987
Vote_153	**0.05**	0.38	0.08	244
Vote_44	10.95	29.08	**2.27**	259,387
Vote_22	47.40	49.14	**3.56**	505,030
Anneal_780	0.05	0.39	**0.03**	104
Anneal_731	**0.18**	0.60	0.22	2,597
Anneal_650	5.70	1.48	**1.05**	47,220
Chess_3037	0.09	1.90	**0.08**	536
Chess_2557	**0.92**	7.64	1.82	191,656
Chess_1918	70.94	254.87	**28.28**	4,633,764
Mushroom_4062	0.09	30.57	**0.06**	109
Mushroom_1747	0.42	72.88	**0.27**	2,215
Mushroom_813	3.45	205.46	**0.87**	11,405
Connect_66882	0.62	828.00	**0.07**	70
Connect_63504	**1.10**	863.17	3.96	51,754
Connect_60802	**7.28**	1107.33	19.82	322,838
T10_500	**0.53**	OOM	5.96	532
T10_100	42.58	OOM	**24.62**	147,531
T10_20	1131.39	OOM	**285.70**	256,896
T40_1600	**3.54**	OOM	14.01	648
T40_1300	**8.97**	OOM	26.86	101,588
T40_100	TO	OOM	TO	$> 10^9$
Pumsb_47085	1.44	OOM	**0.19**	277
Pumsb_43651	**1.99**	OOM	8.47	69,222
Pumsb_39237	**71.96**	OOM	370.58	3,668,125
Retail_441	**2.07**	OOM	3.47	37
Retail_265	**2.01**	OOM	10.64	75
Retail_89	**12.67**	OOM	156.86	250

TO**: timeout** OOM**: out-of-memory**

4.8.8 Mining Constrained Association Rules.

In this section, we present experiments that show the expressiveness of our CP approach in taking into account user's constraints, and the power of CP to solve such combinatorial problems. We illustrate with mandatory/forbidden-item constraints and with cardinality constraints on the body or the head of the rule as a whole (see Section 4.6). We compare our CP approach with SAT and ECLAT-Z-PP (ECLAT-Z with a post-processing step filtering out the rules not satisfying the user's constraints). We selected the instances having more than one million ARs in Table 4.5 (i.e., $\#Tot \geq 10^6$).

Mining rules with constraints on items. The user can ask for rules with some specific items in the rule, the body and/or the head. In the same way, a user can ask for rules not involving a particular set of items. For instance, we may want to extract rules between electronics and cleaning items only. Or to extract rules outside food items. We perform an experiment on the following user query:

> Q_1 : *Given two sets of items $\mathcal{F}$ and $\mathcal{M}$, extract ARs not containing $\mathcal{F}$ in the body, and containing $\mathcal{M}$ in the head.*

Q_1 can easily be expressed in CP with the following constraints:

$$\text{CP-RULE}_{\mathcal{D},s,c}(x, y, z) \wedge Forb_{\mathcal{F}}(x) \wedge Mand_{\mathcal{M}}(y)$$

For our experiment, we generate sets $\mathcal{F}$ and $\mathcal{M}$ of the same size. We randomly select a pair of items and we put one in $\mathcal{F}$ and one in $\mathcal{M}$. We repeat the process until the number of solutions to Q_1 falls under a very low threshold. Table 4.7 reports the penultimate step, that is, the $\mathcal{F}$ and $\mathcal{M}$ where the number of solutions is the smallest above 10. We made an exception with `T10_20`, that we will use to illustrate what happens on an instance with no solution. Table 4.7 reports the CPU time of ECLAT-Z-PP, SAT and CP on Q_1. We also report the number of solutions to Q_1 (#TOT).

The main observation is that the declarative approaches SAT and CP outperform ECLAT-Z-PP. ECLAT-Z-PP extracts all ARs and filters out the rules the user is not asking for. For `Anneal_80`, ECLAT-Z-PP spends 14 minutes to extract and save more than 84 million ARs. The post-processing step spends 10 minutes to filter out this huge number of ARs and to return the only 21 solutions of Q_1. With SAT and CP, the 21 solutions are returned in less than one second.

For the SAT approach, the forbidden and mandatory constraints can easily be encoded with monomials (i.e., unit clauses) that reduce the search space and speed-up the resolution. However, when the size of the dataset increases, the performance of SAT decreases. SAT cannot solve `T40_100` within the time out. In addition, SAT faces again the problem of the size of the formulas and reaches an out-of-memory state on `Pumsb_39237`.

Table 4.7 – ECLAT-Z-PP *vs* SAT *vs* CP *on* Q_1 *(time in seconds)*

Instances	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{M}\rvert$	ECLAT-Z-PP	SAT	CP	#TOT
Zoo_5	11	11	535.69	**0.02**	**0.02**	15
Vote_22	7	7	38.57	0.16	**0.04**	14
Anneal_650	12	12	1454.23	0.19	**0.01**	21
Chess_1918	9	9	292.59	0.54	**0.07**	32
Mushroom_813	10	10	263.93	19.82	**0.02**	21
Connect_60802	9	9	61.04	24.22	**0.02**	19
T10_20	10	10	84.04	TO	**0.02**	0
T40_100	10	10	TO	TO	**0.06**	19
Pumsb_39237	12	12	765.67	OOM	**0.02**	23

TO**: timeout** OOM**: out-of-memory**

On the instance with no solution T10_20, SAT cannot prove that no solution exists within the timeout. ECLAT-Z-PP spends more than 1 minute to extract more than 1 million useless solutions and filter them out. CP proves the absence of solutions in $0.02s$.

Mining rules with cardinality constraints. The user can also ask for rules with particular sizes of body or head. We performed an experiment with the following query:

> Q_2 : *Extract ARs with a body not exceeding a size of* ub *and a head of a minimum size of* lb*.*

Q_2 can easily be expressed in CP with:

$$\text{CP-RULE}_{\mathcal{D},s,c}(x, y, z) \wedge atMost_{ub}(x) \wedge atLeast_{lb}(y)$$

For each instance, we select a lower-bound lb and an upper-bound ub in order to have at least 10 solutions to Q_2. For that, we use two scenarios. The first scenario starts by selecting the largest value for lb for which there exist at least 10 ARs with a head of minimum size lb. Then, we take for ub the size of the smallest body for which there still exist at least 10 solutions. The second scenario selects the smallest value for ub for which there exist at least 10 ARs with a body not exceeding this value. Then, we take for lb the size of the largest head for which there still exist at least 10 solutions. Again, for T10_20, we select lb and ub in order to illustrate the case of no solution. Table 4.8 compares ECLAT-Z-PP, SAT and CP acting on the query Q_2. We report the CPU time, in seconds, for each instance. We also report the total number of solutions (#TOT).

We observe that CP wins on all instances. SAT reaches the one hour timeout on two instances whereas these two instances are solved by CP in less than 10 seconds. Once again, SAT faces an out-of-memory state on Pumsb_39237.

When comparing CP to ECLAT-Z-PP we can observe that for the instances having more than 10 million ARs we have a significant gap in terms of CPU time. CP on Anneal_650

Table 4.8 – ECLAT-Z-PP *vs* SAT *vs* CP *on* Q_2 *(time in seconds)*

Instances	ub	lb	ECLAT-Z-PP	SAT	CP	#TOT
Zoo_5	2	11	479.26	3.92	**0.36**	27
Zoo_5	1	9	491.48	0.17	**0.06**	12
Vote_22	4	8	37.69	282.25	**0.66**	13
Vote_22	1	2	38.49	1.41	**0.05**	23
Anneal_650	2	13	1567.48	1.14	**0.26**	76
Anneal_650	1	12	1622.19	0.53	**0.15**	73
Chess_1918	2	9	280.60	2.17	**0.20**	20
Chess_1918	1	8	284.22	1.07	**0.08**	24
Mushroom_813	1	11	249.00	47.52	**0.07**	14
Connect_60802	1	11	61.80	30.41	**0.26**	12
T10_20	1	11	84.47	TO	**5.44**	0
T40_100	1	11	TO	TO	**8.33**	39
Pumsb_39237	1	12	741.49	OOM	**0.34**	32

TO**: timeout** OOM**: out-of-memory**

returns the 73 solutions in less than one second, whereas ECLAT-Z-PP spends 13 minutes on the post-processing step.

For the instance with no solution T10_20, SAT cannot prove there is no solution before the time out. ECLAT-Z-PP needs a post-processing of more than 1 minute to deal with the 1 million extracted ARs. CP proves there is no solution in less than 6 seconds.

We add a new cardinality constraint to Q_2. In addition to Q_2 with user's constraints on the body and the head of a rule, the user can also ask for rules of a given size:

> Q_3 : *Extract ARs of a size of* v *with a body of size at most* ub *and a head of size at least* lb*.*

Q_3 is expressed using constraints as follows:

$$\text{CP-RULE}_{\mathcal{D},s,c}(x,y,z) \wedge Exactly_v(z) \\ \wedge atMost_{ub}(x) \wedge atLeast_{lb}(y)$$

As for Q_2, we select lb, v and ub in order to have at least 10 solutions. Again, for T10_20, we select lb, v and ub in order to illustrate the case of no solution.

Table 4.9 compares ECLAT-Z-PP, SAT and CP acting on the query Q_3. We report the CPU time, in seconds, for each instance. We also report the total number of solutions (#TOT).

Again, the results reported on Q_3 strengthen our previous observations. CP wins on all instances. SAT reaches the one hour timeout on two instances and an out-of-memory state on Pumsb_39237.

ECLAT-Z needs a post processing of between 1 and 18 minutes to return the only few interesting rules. ECLAT-Z is not able to solve the problem `T40_100` due a timeout. Our CP approach solves the problems in only between 0.13 and 4.37 seconds.

Again, For the instance with no solution `T10_20`, SAT cannot prove there is no solution before the time out. ECLAT-Z-PP needs a post-processing of more than 1 minute to filter all the 1 million extracted ARs. CP proves there is no solution in less than 4 seconds.

Table 4.9 – ECLAT-Z-PP *vs* SAT *vs* CP *on* Q_3 *(time in seconds)*

Instances	v	ub	lb	ECLAT-Z-PP	SAT	CP	#TOT
`Zoo_5`	15	3	12	479.92	6.61	**0.16**	17
`Vote_22`	13	5	7	38.49	756.95	**1.50**	18
`Anneal_650`	13	1	11	1542.12	0.79	**0.13**	69
`Chess_1918`	12	3	9	280.60	6.63	**0.53**	14
`Mushroom_813`	11	1	10	250.56	48.97	**0.05**	77
`Connect_60802`	12	1	11	61.22	24.22	**0.23**	12
`T10_20`	18	1	14	83.96	TO	**3.46**	0
`T40_100`	12	1	11	TO	TO	**4.37**	37
`Pumsb_39237`	13	1	12	761.31	OOM	**0.34**	30

TO**: timeout** OOM**: out-of-memory**

4.8.9 Mining Constrained MNRs.

As pointed out in Section 4.7, anti-monotone constraints on the body and monotone constraints on the head can be injected in our model for mining MNRs without problem. In this section we compare the tree approaches in taking in consideration constraints in addition to the MNR property. We want to mine MNRs w.r.t. Q_4.

> Q_4 : *Extract MNRs with a body not exceeding a size of* ub *and a head of a minimum size of* lb*.*

Q_4 is expressed using constraints as follows:

$$\text{MNRULE}_{\mathcal{D},s,c}(x,y,z) \wedge atMost_{ub}(x) \wedge atLeast_{lb}(y)$$

For Q_4, we selected the instances having more than one million MNRs in Table 4.6 (i.e., $\#Tot \geq 10^6$). The only exception is `Chess_1918` for which CP already outperforms ECLAT-Z for mining MNRs. Adding constraints on that instance would only confirm the outperformance of CP.

For every instance, lb and ub are selected in order to have:
- More than 10 solutions.
- Between 1 and 9 solutions.

- No solution.

Table 4.10 compares ECLAT-Z-PP, SAT and CP acting on the query Q_4. We report the CPU time, in seconds, for each instance. We also report the total number of solutions (#TOT).

Despite the hardness of the instance `T40_100`, our CP approach is able to extract constrained MNRs in less than 10 seconds. ECLAT-Z needs to extract all MNRs to filter inconsistent ones as a post processing step. ECLAT-Z is not able to solve the original problem and does not return any solution within the allocated time.

On `Pumsb_39237`, ECLAT-Z spends more than one minute to extract more than 3 million MNRs to finally filter most of them in a post processing step. CP solve the three instances in less than 1 second. Once again, SAT faces an out-of-memory state on `Pumsb_39237`. This time also on `T40_100`.

Table 4.10 – ECLAT-Z-PP *vs* SAT *vs* CP *on* Q_4 *(time in seconds)*

Instances	ub	lb	ECLAT-Z-PP	SAT	CP	#TOT
`T40_100`	1	11	TO	OOM	**9.53**	33
`T40_100`	1	12	TO	OOM	**9.45**	2
`T40_100`	1	13	TO	OOM	**9.02**	0
`Pumsb_39237`	1	12	157.70	OOM	**0.24**	12
`Pumsb_39237`	1	13	154.24	OOM	**0.28**	2
`Pumsb_39237`	1	14	154.01	OOM	**0.35**	0

TO**: timeout** OOM**: out-of-memory**

4.9 Conclusion

In this chapter, we have introduced a full constraint programming model for mining association rules. For this model, we introduced a new global constraint CONFIDENT for ensuring the confidence of a rule. We proved that enforcing domain consistency on CONFIDENT is NP-hard, ruling out the possibility of a polynomial domain consistency propagator for CONFIDENT (unless $P = NP$). Thus, we proposed for CONFIDENT two weak propagators, one that uses a filtering rule and the other that uses the global constraint COVERSIZE for a decomposition. We have shown that our CP model can easily be extended to take in consideration user's constraints such as mandatory or forbidden items constraints, cardinality constraints, etc. We proposed an extension of our CP model to extract minimal non-redundant rules. For this, we have introduced a new global constraint GENERATOR for mining generators. This constraint can be used to model and solve various itemset mining problems. We proposed a polynomial propagator achieving domain consistency for GENERATOR. We gave a discussion on combining constraints with minimal non-redundant rule. We proved that anti-monotone constraints on the body and monotone constraints on the head are safe on our model.

We have empirically evaluated our CP approach. The experiments showed the performance of our constraints and models comparing to existing approaches. For mining association rules, our CP approach significantly outperforms the existing declarative approach i.e. the SAT approach. That is specially on huge datasets where the SAT approach suffers from a memory issue. Comparing to the specialized algorithm ECLAT-Z, our CP approach performs well and becomes even better than ECLAT-Z for mining minimal non-redundant rules and constrained rules. It is worthy noticing also that our CP approach returns solutions on the fly, and returns the first rules in seconds as a result. On the other hand, ECLAT-Z proceeds in two steps and it is not able to return any solution before the first step has finished.

V

Chapter 5: Constraint Programming for Mining Borders of Frequent Itemsets

5.1 Introduction

For many data mining problems, the frequency represents an important metric. Given a frequency threshold s over a transaction dataset, frequent itemsets are those present in at least s transactions. Itemsets present in less than s transactions are called infrequent (or rare). The number of frequent/infrequent itemsets can be huge, making it hard even to print the result. Hence, we often reduce the problem of mining frequent or infrequent itemsets to the problem of mining the *borders*. The *positive* border is the set of frequent itemsets with only infrequent supersets, i.e. Maximal Frequent Itemsets (MFIs). The *negative* border is the set of infrequent itemsets with only frequent subsets, i.e. Minimal Infrequent Itemsets (MIIs). The subsets of the MFIs and the supersets of the MIIs represent the frequent and infrequent itemsets, respectively.

In this chapter we propose a generic parameterized model allowing the user to decide which border to mine (i.e., MFIs or MIIs). For this model we need to define two new global constraints: (1) FREQUENTSUBS for mining itemsets having only frequent subsets, and (2) INFREQUENTSUPERS for mining itemsets having only infrequent supersets. We provide a polynomial domain consistency propagator for each of these two constraints. For ensuring the infrequency of an itemset, we propose a new global constraint INFREQUENT with a polynomial domain consistency propagator.

We then address the issue of mining borders in the presence of other constraints. As noticed in (Bonchi and Lucchese [2004]), mining borders under additional constraints can lead to the loss of solutions. This can happen with maximal/minimal borders, but also with closed itemsets or generator itemsets. We prove that it is coNP-hard to find maximal/minimal itemsets among those satisfying a set of additional constraints. This implies that there does not exist any CSP representing the problem of finding maximal/minimal itemsets under additional constraints, unless coNP $\subseteq$ NP. This is an a posteriori justifi-

cation of the "multi-shot" approaches, such as the one presented in (Négrevergne et al. [2013]).

This chapter is organized as follows. In Section 5.2 we give a generic model for mining MFIs or MIIs. We define the new global constraints FREQUENTSUBS and INFREQUENTSUPERS and we present their propagators in Sections 5.3 and 5.4, respectively. In Section 5.5 we propose an implementation of the frequency/infrequency constraints. Section 5.6 analyzes the problem of mining constrained MFIs or MIIs. We present some empirical results in Section 5.7 before concluding.

5.2 A Generic CP Model for Mining Borders

We present a CP model, $\text{MODEL}_{\mathcal{D},s,b}$, for mining MFIs or MIIs. $\text{MODEL}_{\mathcal{D},s,b}$ uses a vector x of n Boolean variables. MODEL uses new global constraints that we present later.

$$\text{MODEL}_{\mathcal{D},s,b}(x) = \begin{cases} \text{FREQUENTSUBS}_{\mathcal{D},s}(x) & (1) \\ \text{INFREQUENTSUPERS}_{\mathcal{D},s}(x) & (2) \\ b \iff \text{FREQUENT}_{\mathcal{D},s}(x) & (3) \end{cases}$$

The role of each type of constraint is the following:
(1) is a global constraint that holds if and only if the itemset $x^{-1}(1)$ has only frequent subsets w.r.t. s;
(2) is a global constraint that holds if and only if the itemset $x^{-1}(1)$ has only infrequent supersets w.r.t. s;
(3) the Boolean parameter b reifies the global constraint FREQUENT. $\text{FREQUENT}_{\mathcal{D},s}$ holds if and only if $x^{-1}(1)$ is frequent w.r.t. s.

We prove that $\text{MODEL}_{\mathcal{D},s,b}$ is a correct model for mining MFIs or MIIs.

Theorem 5.1. The set of solutions to $\text{MODEL}_{\mathcal{D},s,b}$ corresponds to the set of MFIs if $b = true$, the set of MIIs otherwise.

Proof. We first prove that MFIs and MIIs always satisfy constraints (1) and (2). All supersets of an MFI are infrequent, thus satisfying (2). Because an MFI is frequent, all its proper subsets are frequent, thus satisfying (1). An MII has only frequent subsets, thus satisfying (1). Because an MII is infrequent, it has only infrequent supersets, thus satisfying (2). Hence, all MFIs and all MIIs satisfy constraints (1) and (2).

We now prove that a solution of $\text{MODEL}_{\mathcal{D},s,b}$, can only be an MFI or an MII depending on the value of b. A solution of $\text{MODEL}_{\mathcal{D},s,b}$ can only be frequent (i.e., $b = true$) or infrequent (i.e., $b = false$). If it is frequent then it is an MFI (see Definition 1.1). Otherwise it is an MII (see Definition 1.4). □

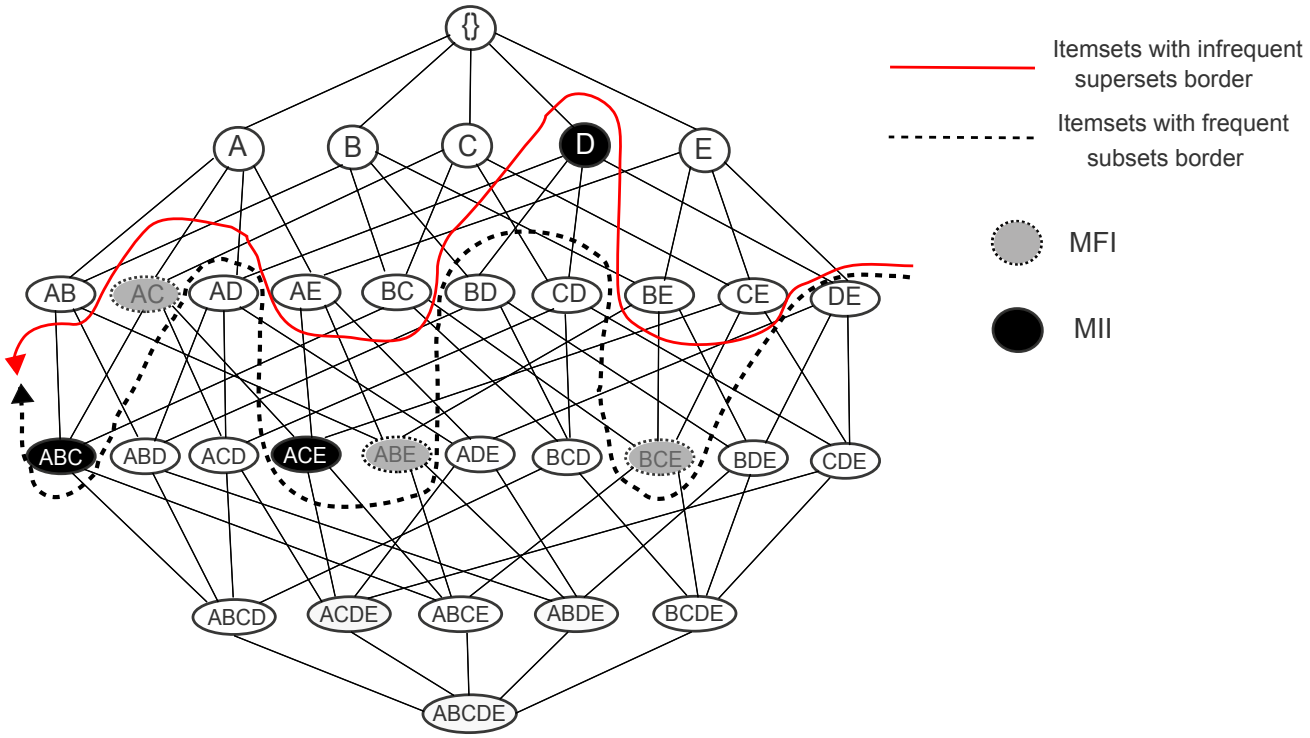


Figure 5.1 – *The powerset lattice of the dataset in Table 1.1 with borders of* FREQUENTSUBS *and* INFREQUENTSUPERS *($s = 3$).*

Example 5.1. The lattice in Figure 5.1 displays the set of itemsets satisfying the constraints FREQUENTSUBS and INFREQUENTSUPERS on the dataset of Table 1.1 with $s = 3$. The intersection between both sets is the set MFIs∪MIIs.

We can then use MODEL$_{\mathcal{D},s,b}$ to mine either MFIs or MIIs by simply setting b to true or false. In the next sections, we define the global constraints FREQUENTSUBS and INFREQUENTSUPERS and we propose propagators.

5.3 The Global Constraint FREQUENTSUBS

In this section we present the new global constraint FREQUENTSUBS. FREQUENTSUBS is used to mine itemsets that have only frequent subsets.

Definition 5.1 (FREQUENTSUBS)**.** Let x be a vector of Boolean variables, s a frequency threshold and $\mathcal{D}$ a dataset. The global constraint FREQUENTSUBS$_{\mathcal{D},s}(x)$ holds if and only if $\forall P \subsetneq x^{-1}(1)$, $freq(P) \geq s$.

Algorithm 2: Propagator for FREQUENTSUBS

```
1  In: s: frequency threshold;
2  InOut: x = {x_1 ... x_n}: Boolean item variables;
3  begin
4      cover ← cover(x^{-1}(1));
5      foreach j ∈ x^{-1}(1) do
6          cov[j] ← cover(x^{-1}(1) \ {j});
7          if |cov[j]| < s then return failure ;
8      if |cover| < s then
9          foreach i ∈ x^{-1}(*) do
10             dom(x_i) ← dom(x_i) \ {1};
11     foreach i ∈ x^{-1}(*) do
12         if |cover ∩ cover(i)| < s then
13             foreach j ∈ x^{-1}(1) do
14                 if |cov[j] ∩ cover(i)| < s then
15                     dom(x_i) ← dom(x_i) \ {1}; break;
```

Example 5.2. Consider the dataset in Table 1.1 with $s = 3$.
FREQUENTSUBS$_{\mathcal{D},3}([0,1,0,0,1])$ is satisfied because BE has only frequent subsets w.r.t. s.
FREQUENTSUBS$_{\mathcal{D},3}([1,0,0,1,0])$ is not satisfied because AD has an infrequent subset, i.e. D.

Algorithm. The propagator for the global constraint FREQUENTSUBS is presented in Algorithm 2. Algorithm 2 takes as input the variables x and the frequency threshold s. Algorithm 2 starts by computing the cover of the itemset $x^{-1}(1)$ and stores it in `cover` (line 4). Then, for each item $j \in x^{-1}(1)$, Algorithm 2 computes the cover of the subset $x^{-1}(1) \setminus \{j\}$, and stores it in `cov[j]` (line 6). If $x^{-1}(1) \setminus \{j\}$ is infrequent then $x^{-1}(1)$ is not a solution and we return a failure (line 7). Algorithm 2 must then remove items i that cannot belong to a solution containing $x^{-1}(1)$. To do that, we first test if the itemset $x^{-1}(1)$ is infrequent (line 8). If so, we remove 1 from $dom(x_i)$ for all $i \in x^{-1}(*)$ (lines 9-10) because the itemset $x^{-1}(1) \cup \{i\}$ has an infrequent subset ($x^{-1}(1)$) for every i in $x^{-1}(*)$. Otherwise, for every item i in $x^{-1}(*)$ we test if the itemset $x^{-1}(1) \cup \{i\}$ is infrequent (line 12). If so, it could have infrequent subsets. Thus, for every item j in $x^{-1}(1)$, we test if the size of the cover of $x^{-1}(1) \cup \{i\} \setminus \{j\}$ (i.e., `cov[j]` $\cap\, cover(i)$) is less than s (line 14). If so, we remove i from the possible items, that is, we remove 1 from $dom(x_i)$ and break the loop (line 15).

Example 5.3. Consider the dataset in Table 1.1 with $s = 3$. Given the partial assignment $x^{-1}(1) = \{D\}$, $x^{-1}(0) = \emptyset$ and $x^{-1}(*) = \{A, B, C, E\}$, Algorithm 2 filters 1 from $dom(x_A)$ because the itemset AD has an infrequent subset, i.e. D ($freq(D) = 1 < s$).

Theorem 5.2. The propagator in Algorithm 2 enforces domain consistency on the constraint FREQUENTSUBS.

Proof. We first prove that if FREQUENTSUBS admits a solution, $x^{-1}(1)$ is necessarily one of them. Suppose there is a solution and $x^{-1}(1)$ is not one of them. This means that there exists a superset of $x^{-1}(1)$ which has all its subsets frequent. $x^{-1}(1)$ is one of these subsets. Thus, all subsets of $x^{-1}(1)$ are frequent and $x^{-1}(1)$ is solution too, which contradicts the assumption. We now prove that Algorithm 2 returns failure if and only if FREQUENTSUBS does not admit any solution. We know that FREQUENTSUBS has no solution if and only if $x^{-1}(1)$ is not solution. $x^{-1}(1)$ is not solution if and only if it has an infrequent subset $x^{-1}(1) \setminus \{j\}$ for some j in $x^{-1}(1)$. In such a case the test in line 7 is true and failure is returned. If FREQUENTSUBS admits solutions, $x^{-1}(1)$ is a support for $x_i = 0$, for all $i \in x^{-1}(*)$. As a result, Algorithm 2 does not need to check consistency of value 0 for any variable.

We now prove that Algorithm 2 prunes value 1 from $dom(x_i)$ exactly when i cannot belong to a solution containing $x^{-1}(1)$. Suppose value 1 of x_i is pruned by Algorithm 2. This means that the test in line 8 (or line 14) was true, that is, there exists a subset of $x^{-1}(1) \cup \{i\}$ which is infrequent. Thus, by definition, $x^{-1}(1) \cup \{i\}$ does not belong to any solution. Suppose now that value 1 of x_i is not pruned. From lines 8 and 14, we deduce that there does not exist any infrequent subset of $x^{-1}(1) \cup \{i\}$. Thus $x^{-1}(1) \cup \{i\}$ is a solution and value 1 of x_i is domain consistent. □

Theorem 5.3. Given a transaction dataset $\mathcal{D}$ of n items and m transactions, Algorithm 2 has an $O(n^2 \times m)$ time complexity.

Proof. Computing the size of the cover of an itemset is in $O(n \times m)$. Line 5 is called at most n times, leading to a time complexity in $O(n^2 \times m)$. The time complexity of lines 8-10 is bounded above by n. The test at line 13 is done at most n^2 times. The cover $x^{-1}(1) \cup \{i\} \setminus \{j\}$ at line 13 is computed in $O(m)$ thanks to the `cov` data structure. Thus, the time complexity of lines 9-14 is bounded above by $n^2 \times m$. As a result, Algorithm 2 has an $O(n^2 \times m)$ time complexity. □

5.4 The Global Constraint INFREQUENTSUPERS

In this section we present the new global constraint INFREQUENTSUPERS. INFREQUENTSUPERS is used to mine itemsets that have only infrequent supersets.

Definition 5.2 (INFREQUENTSUPERS)**.** Let x be a vector of Boolean variables, s a frequency threshold and $\mathcal{D}$ a dataset. The global constraint INFREQUENTSUPERS$_{\mathcal{D},s}(x)$ holds if and only if $\forall P \supsetneq x^{-1}(1)$, $freq(P) < s$.

Example 5.4. Consider the dataset in Table 1.1 with $s = 3$.
INFREQUENTSUPERS$_{\mathcal{D},3}([1,1,1,0,1])$ is satisfied because $ABCE$ has only infrequent supersets w.r.t. s. INFREQUENTSUPERS$_{\mathcal{D},3}([1,0,0,0,1])$ is not satisfied because AE has a frequent superset, i.e. ABE.

Algorithm 3: Propagator for INFREQUENTSUPERS

```
1  In: s: frequency threshold;
2  InOut: x = {x_1 ... x_n}: Boolean item variables;
3  begin
4      cover ← cover(x^{-1}(1) ∪ x^{-1}(*));
5      if |cover| ≥ s then
6          foreach j ∈ x^{-1}(0) do
7              if |cover ∩ cover(j)| ≥ s then
8                  return failure;
9      foreach i ∈ x^{-1}(*) do
10         cover2 ← cover(x^{-1}(1) ∪ x^{-1}(*) \ {i});
11         if |cover2| ≥ s then
12             foreach j ∈ (x^{-1}(0) ∪ {i}) do
13                 if |cover2 ∩ cover(j)| ≥ s then
14                     dom(x_i) ← dom(x_i) \ {0}; break;
```

Algorithm. The propagator for the global constraint INFREQUENTSUPERS is presented in Algorithm 3. Algorithm 3 takes as input the variables x and the frequency threshold s. Algorithm 3 starts by computing `cover`, the cover of the largest possible itemset, $x^{-1}(1) \cup x^{-1}(*)$ (line 4). If such an itemset has a frequent superset (line 7) then no itemset containing $x^{-1}(1)$ can be a solution and we return failure (line 8). Algorithm 3 must then remove value 0 from $dom(x_i)$ if the absence of the item i can lead to an itemset that has a frequent superset. To do that, for every free item i (line 9), Algorithm 3 computes `cover2`, the cover of the largest itemset not containing the item i, i.e. $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}$ (line 10). If $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}$ is frequent (line 11), it could have a frequent superset. Thus, for every item j in $x^{-1}(0) \cup \{i\}$ we test whether the itemset $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\} \cup \{j\}$ is frequent (line 13). If so, we remove 0 from $dom(x_i)$ and break the loop (line 14).

Example 5.5. Consider the dataset in Table 1.1 with $s = 3$. Given the partial assignment $x^{-1}(1) = \{A\}$, $x^{-1}(0) = \{B, D\}$ and $x^{-1}(*) = \{C, E\}$, Algorithm 2 filters 0 from $dom(x_C)$ because the itemset AE has a frequent superset i.e. ABE ($freq(ABE) = 3 \geq s$).

Theorem 5.4. The propagator in Algorithm 3 enforces domain consistency on the constraint INFREQUENTSUPERS.

Proof. We first prove that if INFREQUENTSUPERS admits a solution, $x^{-1}(1) \cup x^{-1}(*)$ is necessarily one of them. Suppose there is a solution and $x^{-1}(1) \cup x^{-1}(*)$ is not one of them. This means that there exists a subset of $x^{-1}(1) \cup x^{-1}(*)$ which has all its supersets infrequent. $x^{-1}(1) \cup x^{-1}(*)$ is one of these supersets. Thus, all supersets of $x^{-1}(1) \cup x^{-1}(*)$ are infrequent and $x^{-1}(1) \cup x^{-1}(*)$ is solution too, which contradicts the assumption. We now prove that Algorithm 3 returns failure if and only if INFREQUENTSUPERS does not admit

any solution. We know that if INFREQUENTSUPERS has solutions, $x^{-1}(1) \cup x^{-1}(*)$ is one of them. $x^{-1}(1) \cup x^{-1}(*)$ is not solution if and only if it has a frequent superset. In such a case the test in line 7 is true for some j in $x^{-1}(0)$ and failure is returned. If INFREQUENTSUPERS has solutions, $x^{-1}(1) \cup x^{-1}(*)$ is a support for $x_i = 1$, for all $i \in x^{-1}(*)$. As a result, Algorithm 3 does not need to check consistency of value 1 for any variable.

We now prove that Algorithm 3 prunes 0 from $dom(x_i)$ exactly when i belongs to all solutions containing $x^{-1}(1)$. If the test in line 13 is true this means that with $x_i = 0$ we get itemsets having at least a frequent superset. Hence, 0 must be pruned from $dom(x_i)$. Suppose now that the test in line 13 is false for all j. $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}$ has only infrequent supersets and $x_i = 0$ is domain consistent. □

Theorem 5.5. Given a transaction dataset $\mathcal{D}$ of n items and m transactions, Algorithm 3 has an $O(n^2 \times m)$ time complexity.

Proof. Computing the cover of $x^{-1}(1) \cup x^{-1}(*)$ in line 4 is in $O(n \times m)$. The loop in line 6 computes the cover of $x^{-1}(1) \cup x^{-1}(*) \cup \{j\}$ in $O(m)$ using the already computed cover of $x^{-1}(1) \cup x^{-1}(*)$ (i.e., `cover`). As a result the worst case time complexity of lines 4-8 is in $O(n \times m)$. Similarly, the time complexity of lines 10-14 is bounded above by $n \times m$. Lines 10-14 are called at most n times, leading to a time complexity in $O(n^2 \times m)$. □

The Duality Between Propagators

It is worth noticing the duality between Algorithm 2 and Algorithm 3. Algorithm 2 removes value 1 from $dom(x_i)$ if including the item i leads to an itemset having an infrequent subset. Algorithm 3 removes value 0 from $dom(x_i)$ if excluding the item i necessarily leads to itemsets having a frequent superset. This duality is not totally obvious at a first glance because Algorithm 2 requires the `cov` data structure to have its good time complexity whereas Algorithm 3 does not need it. The reason is that Algorithm 2 computes the covers of *subsets* that we cannot derive from the cover of their supersets whereas Algorithm 3 computes covers of *supersets* that can be obtained by simple bitwise operation on their subsets.

On the Conjunction of FREQUENTSUBS and INFREQUENTSUPERS

To mine itemsets that have frequent subsets and infrequent supersets at the same time we combine the constraints FREQUENTSUBS and INFREQUENTSUPERS. One may ask if the propagators of FREQUENTSUBS and INFREQUENTSUPERS enforce DC on the constraint network defined by the conjunction of FREQUENTSUBS and INFREQUENTSUPERS. The answer is no and we show this in the following example.

Example 5.6. Consider the dataset in Table 5.1 with $s = 1$. With the partial assignment $x^{-1}(1) = \{D\}$, $x^{-1}(0) = \{A\}$ and $x^{-1}(*) = \{B, C, E\}$, neither the propagator of FRE-

Table 5.1 – *Dataset*

trans.	Items
t_1	$A\ B\ \ \ E$
t_2	$\ \ B\ \ D\ \ $
t_3	$\ \ B\ C\ \ \ $
t_4	$A\ B\ \ D\ \ $
t_5	$A\ \ \ C\ \ \ $
t_6	$\ \ B\ C\ \ \ $
t_7	$A\ \ \ C\ \ \ $
t_8	$A\ B\ C\ \ \ E$
t_9	$A\ B\ C\ \ \ $

QUENTSUBS nor the propagator of INFREQUENTSUPERS can filter any value. However, value 1 for $dom(x_B)$ is not domain consistent. That is because with $x_B = 1$ we get the partial assignment $x^{-1}(1) = \{B, D\}$, $x^{-1}(0) = \{A\}$ and $x^{-1}(*) = \{C, E\}$. Value 1 is filtered by the propagator of FREQUENTSUBS from $dom(x_C)$ and $dom(x_E)$ leading to the complete assignment $x^{-1}(1) = \{B, D\}$, $x^{-1}(0) = \{A, C, E\}$ and $x^{-1}(*) = \emptyset$. BD does not satisfy the constraint INFREQUENTSUPERS (its superset ABD is frequent) and value 1 is not domain consistent for $dom(x_B)$.

5.5 Implementation of the Frequency Constraint

For encoding constraint (3) in MODEL (see Section 5.2), we may use COVERSIZE with $p \geq s$ to ensure the frequency (when $b = true$) and with $p < s$ to ensure the infrequency (when $b = false$). The propagator of COVERSIZE enforces DC on the frequency, but it does not on the infrequency (see Section 3.5). Hence, we propose a new global constraint INFREQUENT for mining infrequent itemsets. We provide INFREQUENT with a filtering rule that enforces DC.

Definition 5.3 (INFREQUENT)**.** Let x be a vector of Boolean variables, s the frequency threshold and $\mathcal{D}$ a dataset. The global constraint $\text{INFREQUENT}_{\mathcal{D},s}(x)$ holds if and only if $x^{-1}(1)$ is infrequent w.r.t. s, i.e. $freq(x^{-1}(1)) < s$.

Example 5.7. Consider the dataset in Table 1.1 with $s = 3$. $\text{INFREQUENT}_{\mathcal{D},3}([0, 0, 0, 1, 0])$ is satisfied because D is infrequent w.r.t. s. $\text{INFREQUENT}_{\mathcal{D},3}([1, 1, 0, 0, 0])$ is not satisfied because AB is frequent w.r.t. s.

Proposition 5.1 (Filtering Rule for INFREQUENT). For every $i \in x^{-1}(*)$, if $freq(x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}) \geq s$ then remove 0 from $dom(x_i)$.

Example 5.8. Consider the dataset in Table 1.1 with $s = 3$. Given the partial assignment $x^{-1}(1) = \emptyset$, $x^{-1}(0) = \{A\}$ and $x^{-1}(*) = \{B, C\}$, the filtering rule in Proposition 5.1 filters 0 from $dom(x_C)$ because the itemset B is frequent w.r.t. s. Note that this value is not filtered

by the propagator of COVERSIZE (see Example 3.7).

Theorem 5.6. The filtering rule presented in Proposition 5.1 enforces domain consistency on the constraint INFREQUENT.

Proof. We first prove that the value 1 for a variable x_i such that $i \in x^{-1}(*)$ always belongs to a solution and hence cannot be pruned by domain consistency. For this we prove that if INFREQUENT admits a solution, $x^{-1}(1) \cup x^{-1}(*)$ is necessary one of them. Suppose that there is a solution and $x^{-1}(1) \cup x^{-1}(*)$ is not one of them. This means that there exits a subset of $x^{-1}(1) \cup x^{-1}(*)$ which is infrequent. Thus, $x^{-1}(1) \cup x^{-1}(*)$ is infrequent too which contradicts the assumption. As a result, value 1 cannot be pruned from $dom(x_i)$.

We now prove that the filtering rule in Proposition 5.1 prunes 0 from $dom(x_i)$ exactly when i must be added to the itemset $x^{-1}(1)$. If the test in Proposition 5.1 is true this means that by excluding the item i we get only frequent itemsets. Hence, 0 must be pruned from $dom(x_i)$. Suppose now that the test in Proposition 5.1 is false, in this case $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}$ is an infrequent itemset with $x_i = 0$. Thus 0 is domain consistent. □

5.6 On Constrained Borders

As pointed out in (Bonchi and Lucchese [2004]), constraints can interfere with closeness (or maximality) when they are not monotone. Likewise, constraints can interfere with minimality when they are not anti-monotone. Hence, existing "one-shot" CP approaches can miss solutions because they look for maximal/minimal itemsets that in addition satisfy constraints (e.g., CLOSEDPATTERN with non monotone constraints), whereas we are usually interested in itemsets that are maximal (or minimal) among those satisfying the constraints (see Section 1.6.1). In this section we prove that deciding whether a frequent itemset is maximal or closed or minimal among those satisfying the constraints is coNP-complete. This means that finding maximal/closed/minimal itemsets among those satisfying a set of constraints is coNP-hard. It is a proof that it is not possible to solve this problem using a single CSP (unless coNP $\subseteq$ NP). This validates "multi-shot" approaches (Négrevergne et al. [2013]).

Theorem 5.7. Given a dataset $\mathcal{D}$ on a set of items $\mathcal{I}$ and a set C of user's constraints, deciding whether an itemset is maximal/closed among those satisfying C is coNP-complete.

Proof. Membership. Given an itemset P, a witness to its non maximality/closeness is an itemset $P' \supsetneq P$ that is frequent, satisfies C, and in the case of closeness has the same frequency as P. Checking that P' is frequent and checking that it has the same frequency as P is linear in $|\mathcal{D}|$. Checking that P' satisfies C requires the polynomial check of the C constraints. Hence, the "no" answer admits a polynomial certificate, and so deciding maximality or closeness is in coNP.

Completeness. We reduce co3COL, which is coNP-complete, to the problem of deciding whether an itemset is maximal/closed. We want to decide whether a connected graph $G = (V, E)$ is non colorable with three colors. We build the dataset $\mathcal{D}$ on the set $\mathcal{I} = (a_1, \ldots, a_n, b_1, \ldots, b_n, c)$ of items, where $n = |V|$. The pair (a_i, b_i) of items will represent the vertex i in V. $\mathcal{D}$ contains the single transaction $(1, \ldots, 1)$ and the frequency threshold s is set to $|\mathcal{T}|/2$.

As in all CP models for itemset mining, there is a Boolean variable for each item. The standard semantics is that $x_p = 1$ if and only if the item p is in the itemset returned as solution.

For each edge $(i, j) \in E$, a quaternary constraint $c(x_{a_i}, x_{b_i}, x_{a_j}, x_{b_j})$ is put in the set C of user's constraints. The tuples allowed by $c(x_{a_i}, x_{b_i}, x_{a_j}, x_{b_j})$ are $(0000), (0110), (0111), (1001), (1011), (1101), (1110)$. The assignments (01), (10), and (11) for the pair of variables (x_{a_i}, x_{b_i}) represent the three colors for vertex i. $c(x_{a_i}, x_{b_i}, x_{a_j}, x_{b_j})$ accepts the tuple (0000) plus the six tuples representing the six combinations of different colors for the pair of vertices (i, j). Hence, by construction, the tuple $(0, \ldots, 0)$ is solution, and, as soon as a variable x_{a_i} or x_{b_i} is set to 1, it forces all neighbors in E of vertex i to take another color than the color represented by the assignment of (x_{a_i}, x_{b_i}). As a result, deciding whether the itemset $\{c\}$ is maximal/closed among the itemsets satisfying C is equivalent to deciding whether there does not exist any superset of $\{c\}$ satisfying C, which is equivalent to deciding whether G is non 3-colorable. □

Theorem 5.8. Given a dataset $\mathcal{D}$ on a set of items $\mathcal{I}$ and a set C of user's constraints, deciding whether an itemset is minimal/generator among those satisfying C is coNP-complete.

Proof. (Sketch.) Membership is direct adaptation of the proof of membership in Theorem 5.7: A witness of non-minimality/non-generator of an itemset P is a subset that is infrequent (or has the same frequency in case of generator) and satisfies C, which is polynomial to check. Completeness is the dual of the reduction in Theorem 5.7. We reduce co3COL to the problem of deciding whether an itemset is minimal/generator. We build the dataset $\mathcal{D}$ on the set $\mathcal{I} = (a_1, \ldots, a_n, b_1, \ldots, b_n)$ of items with the three transactions $(1, 0, \ldots, 0)$, $(0, \ldots, 0, 1)$ and $(1, \ldots, 1)$. The frequency threshold s is set to $|\mathcal{T}|/2$. For each edge in the graph, we use again a quaternary constraint but the tuple (0000) is replaced by the tuple (1111) and the three colors are represented by the assignments (00), (01), and (10) instead of (01), (10), and (11). By construction, the tuple $(1, \ldots, 1)$ is solution and deciding whether the itemset $(a_1, \ldots, a_n, b_1, \ldots, b_n)$ is minimal/generator among the itemsets satisfying C is equivalent to deciding whether there does not exist any subset of $(a_1, \ldots, a_n, b_1, \ldots, b_n)$ satisfying C, which is equivalent to deciding whether G is non 3-colorable. □

Corollary 5.1. Given a dataset $\mathcal{D}$ on a set of items $\mathcal{I}$ and a set C of user's constraints, finding an itemset that is maximal/closed/minimal/generator among those satisfying C is coNP-hard.

5.7 Experiments

We made several experiments to compare our CP model MODEL$_{\mathcal{D},s,b}$ to the state of the art approaches.

5.7.1 Experimental Protocol

The implementation of MODEL$_{\mathcal{D},s,b}$ and its constraint propagators were carried out in the `Oscar` solver using Scala.[1] The code is publicly available.[2] All experiments were conducted on an Intel core i7, 2.2Ghz with a RAM of 8Gb and a timeout of one hour. MODEL$_{\mathcal{D},s,b}$ is denoted by CP4MFI (resp. CP4MII) when $b = 1$, i.e. mining MFIs (resp. $b = 0$, i.e. mining MIIs). We used the global constraint COVERSIZE to encode the constraint (3) in MODEL$_{\mathcal{D},s,b}$. We enforce frequency by simply adding the constraint $p \geq s$. As for the infrequency, we have assessed the quality of the COVERSIZE encoding of infrequency by comparing to the propagator of the new constraint INFREQUENT.

As an MFI is a closed itemset and an MII is a generator, we enhance propagation by adding, respectively, the global constraints COVERCLOSURE (Schaus et al. [2017]) and GENERATOR (Belaid et al. [2019b]). After a few preliminary tests, we decided to use *smallest item frequency* first as variable ordering heuristic and *largest value* first as value ordering heuristic. We compared CP4MFI to the FP-GROWTH (Grahne and Zhu [2003]) specialized algorithm for extracting MFIs. We used for this the Borgelt's data mining platform.[3] We compared CP4MII to WALKY-G (Szathmary et al. [2012]) for mining MIIs using the CORON data mining platform.[4] We selected several real-sized datasets from the FIMI repository.[5] A dataset is characterized by its name, the number of items $|\mathcal{I}|$, the number of transactions $|\mathcal{T}|$ and its density ρ (see Table 4.1).

5.7.2 The Infrequency

We start by comparing the propagator of COVERSIZE for ensuring the infrequency with the propagator of the new global constraint INFREQUENT.

The number of infrequent itemsets is huge, hence we mine only non-zero infrequent itemsets in this experiment (infrequent itemsets that appear in at least one transaction). For each propagator and each selected instance, Table 5.2 reports the CPU time in seconds, the number of nodes, the number of fails and the number of infrequent itemsets ($\#Sol$). An instance of a given dataset is characterized by its frequency threshold s (e.g., `Zoo_2` denotes the instance of `Zoo` with $s = 2$).

1. bitbucket.org/oscarlib/oscar
2. gite.lirmm.fr/belaid/cp4borders
3. borgelt.net/fpgrowth.html
4. coron.loria.fr
5. fimi.ua.ac.be/data

We observe that the number of additional nodes explored by the propagator of COVERSIZE is marginal comparing to the propagator of INFREQUENT. But the CPU time is better using the propagator of COVERSIZE. Take for instance `Vote_10` where the number of nodes/fails are reduced using the propagator of INFREQUENT. However, the use of the COVERSIZE propagator reduces the CPU time by 4 seconds on `Vote_10`. This is due to the fact that the propagator of COVERSIZE does not compute the cover of $x^{-1}(1) \cup x^{-1}(*) \setminus \{i\}$ in every node but only when $UB(p) = |cover(x^{-1}(1) \cup x^{-1}(*))|$ (see Section 3.5). In the next experiments we use the propagator of COVERSIZE for ensuring the infrequency.

Table 5.2 – COVERSIZE *vs* INFREQUENT *for mining infrequent itemsets (time in seconds)*

Dataset $\lvert\mathcal{I}\rvert \times \lvert\mathcal{T}\rvert$ $\rho(\%)$	s	COVERSIZE			INFREQUENT			$\#Sol$
		Nodes	Fails	Time(s)	Nodes	Fails	Time(s)	
`Zoo`	2	2,604,234	427,622	2.11	**2,117,054**	**184,032**	**2.07**	874,496
36×101	6	3,738,154	222,786	**2.73**	**3,353,040**	**30,229**	2.78	1,646,292
44%	10	3,817,790	109,687	**2.86**	**3,613,174**	**7,379**	3.40	1,799,209
`Vote`	2	17,774,002	1,431,723	**12.61**	**16,366,434**	**727,940**	14.64	7,455,279
48×435	6	21,229,000	727,822	**14.57**	**20,219,334**	**222,989**	18.51	9,886,679
33%	10	21,465,596	455,058	**14.61**	**20,817,530**	**131,025**	18.63	10,277,741

5.7.3 Mining Borders

This experiment compares CP4MFI to FP-GROWTH and CP4MII to WALKY-G for mining MFIs and MIIs. For each approach and each selected instance, Table 5.3 reports the CPU time and the number of MFIs/MIIs. [6] An instance of a given dataset is characterized by its frequency threshold s (e.g., `Zoo_50` denotes the instance of `Zoo` with $s = 50$).

A first observation is that the specialized algorithms FP-GROWTH and WALKY-G follow a completely different approach to extract MFIs and MIIs whereas our CP model can mine a border or the other just by flipping a parameter. When mining MFIs, the main observation that we can draw from Table 5.3 is that, as expected, the specialized algorithm FP-GROWTH performs very well. However, CP4MFI is very competitive too, and even faster on one instance (`Chess_160`). As for MIIs, CP4MII outperforms WALKY-G in 14 instances out of 26. CP4MII reaches once the timeout of one hour. WALKY-G reaches the time out on two instances and an out of memory state on three instances. WALKY-G uses a hash structure for storing frequent generators. Maintaining this data structure can be very expensive, especially on dense datasets. For instance, on the very dense instance `Chess_500`, WALKY-G exhausts the $8Gb$ of memory to store the $50M$ frequent generators. On the moderately dense `Connect_7000` WALKY-G needs to store more than $7.4M$ frequent generators to find the $165K$ MIIs in 163.25 seconds. On these two instances, CP4MII returns the whole set of MIIs in only 97.42 and 99.42 seconds, respectively. On sparse datasets, where we have few frequent generators, WALKY-G is very efficient. On the very sparse dataset `T10`, WALKY-G stores less than $50K$ frequent generators to extract the $698K$ MIIs of `T10_50` in less than 5 seconds whereas CP4MII needs more than 7 minutes.

6. This number is computed by releasing the timeout.

Table 5.3 – FP-GROWTH *vs* CP4MFI *for mining MFI and* WALKY-G *vs* CP4MII *for mining MIIs (time in seconds)*

Dataset $\lvert\mathcal{I}\rvert \times \lvert\mathcal{T}\rvert$ $\rho(\%)$	s	Mining MFIs			Mining MIIs		
		(a)	CP4MFI	#MFIS	(b)	CP4MII	#MIIS
Zoo	50	**0.01**	0.03	32	0.12	**0.04**	111
36×101	9	**0.01**	0.08	200	**0.10**	0.13	875
44%	1	**0.01**	0.09	59	0.27	**0.12**	1,071
Vote	150	**0.01**	0.04	75	0.19	**0.10**	479
48×435	5	**0.08**	0.77	13,787	**0.85**	1.40	37,526
33%	1	**0.13**	0.47	342	1.26	**1.24**	32,067
Anneal	700	**0.01**	0.05	65	0.26	**0.08**	303
93×812	100	**0.42**	1.10	15,889	9.59	**2.06**	77,119
45%	50	**0.61**	1.13	14,296	23.48	**3.41**	86,783
Chess	2,500	**0.01**	0.10	286	**0.10**	0.13	511
$75 \times 3,196$	1,000	**1.04**	4.34	114,382	23.14	**8.38**	152,316
49%	500	**16.60**	25.66	952,812	OOM	**97.42**	1,353,344
	160	233.78	**127.30**	5,784,232	OOM	**923.84**	9,364,262
Mushroom	4,000	**0.01**	0.03	12	0.31	**0.06**	145
$119 \times 8,124$	40	**0.07**	1.09	12,010	**0.86**	2.33	48,111
19%	4	**0.24**	1.56	39,416	**1.95**	4.58	49,046
Connect	55,000	**0.01**	0.51	594	**0.34**	0.48	891
$129 \times 67,557$	7,000	**2.60**	69.03	123,345	163.25	**99.42**	165,198
33%	2,000	**24.40**	342.95	893,826	TO	**885.99**	1,180,278
	676	**102.90**	800.65	3,283,735	TO	**3541.77**	4,221,496
T10	5,000	**0.01**	0.19	10	**0.13**	4.59	905
$1,000 \times 100,000$	1,000	**0.12**	207.89	307	**2.33**	373.01	344,651
1%	50	**0.35**	255.18	12,062	**4.27**	472.82	698,556
Pumsb	48,000	**0.01**	0.09	3	**0.25**	10.18	2,115
$2,113 \times 49,046$	32,000	**0.21**	470.80	17,791	**30.10**	48.55	57,425
3%	17,000	**145.96**	TO	2,403,260	OOM	TO	$> 3 \times 10^6$

(a): FP-GROWTH **(b):** WALKY-G

T10 **= T10I4D100K** TO**= timeout** OOM**= out of memory**

5.7.4 Mining Constrained Borders

In many practical applications, the user asks for itemsets satisfying some additional constraints. We performed experiments that show the strength of our CP approach in taking into account user's constraints. We look for MFIs of minimim size lb and MIIs of maximum size ub. These two queries can easily be expressed in our CP model by adding the cardinality constraints $\sum_{i\in\mathcal{I}} x_i \geq lb$ to CP4MFI, and $\sum_{i\in\mathcal{I}} x_i \leq ub$ to CP4MII. FP-GROWTH includes a filtering step allowing us to specify the minimum size of extracted MFIs. However, WALKY-G does not provide such feature, and extracting MIIs under cardinality constraints is only possible via a post-processing step. Table 5.4 reports the results. We selected the instances having more than one million MFIs and/or MIIs in

Table 5.4 – FP-GROWTH *vs* CP4MFI *for mining constrained MFI and* WALKY-G *vs* CP4MII *for mining constrained MIIs (time in seconds)*

Instance	Mining constrained MFIs				Mining constrained MIIs			
	lb	(a)	CP4MFI	#SOL	ub	(b)	CP4MII	#SOL
Chess_500	25	17.97	**0.30**	0	1	OOM	**0.08**	19
	24	17.60	**0.72**	2	3	OOM	**0.41**	1,962
	21	16.60	**1.65**	2,091	5	OOM	**8.88**	31,591
	17	17.86	**10.58**	171,567	7	OOM	**16.58**	224,172
Chess_160	29	201.55	**1.23**	0	1	OOM	**0.03**	9
	28	224.31	**1.73**	10	3	OOM	**0.62**	2,384
	26	226.77	**3.51**	1,527	4	OOM	**1.31**	13,487
	23	219.82	**16.08**	132,814	6	OOM	**17.44**	186,653
Connect_676	40	117.87	**5.96**	0	1	TO	**0.15**	20
	33	111.95	**69.00**	266	2	TO	**2.53**	1,612
	32	**109.07**	117.17	21,788	4	TO	**22.81**	33,742
	31	**125.62**	171.04	157,793	5	TO	**119.42**	144,007
Pumsb_17000	35	157.28	**1.16**	0	1	OOM	**4.59**	2,033
	30	157.58	**14.96**	21	3	OOM	**11.94**	5,721
	26	153.33	**148.26**	1,663	6	OOM	**254.58**	166,178
	22	**154.52**	3019.25	55,018	8	OOM	**2702.20**	1,388,040

(a): FP-GROWTH **(b):** WALKY-G
TO= **timeout** OOM= **out of memory**

Table 5.3. For each selected instance, we increased lb (resp. decreased ub) and we report the CPU time, in seconds, according to the number of solutions.

The main observation on the performance of FP-GROWTH for extracting constrained MFIs is that its CPU time is almost constant. For instance, extracting more than $5M$ or just 10 MFIs on Chess_160 requires almost the same CPU time. This is explained by the fact that the cardinality constraint in FP-GROWTH has no pruning power. It is just used as a checker. On the contrary, when lb increases, CP4MFI removes more values thanks to constraint propagation, thus drastically reducing the search space. Take for instance Pumsb_17000 When $lb = 22$, CP4MFI extracts the $55K$ MFIs in more than 50 minutes whereas when lb increases to 30, CP4MFI returns the 21 remaining solutions in only 15 seconds. On instances with no solutions, such as Pumsb_17000 with $lb = 35$, CP4MFI proves the absence of solutions in one second whereas FP-GROWTH spends 157.28 seconds.

For WALKY-G, no results are reported because it already has an OOM or TO state before reaching post-processing step. The main observation that we can draw on the behavior of CP4MII is again the strong correlation between the increase of tightness of the cardinality constraint (that is, the decrease of ub) and the drop in CPU time needed to extract the constrained MIIs. The explanation for this good behavior of CP4MII is again the strength of constraint propagation to discard inconsistent values, thus reducing the search space. On Pumsb_17000, when ub decreases from 8 down to 1, CP4MII goes from 45 minutes down to 4.59 seconds to return the set of MIIs ($1.3M$ MIIs for $ub = 8$ and $2K$ MIIs for $ub = 1$).

5.8 Conclusion

In this chapter, we have presented a CP model allowing the mining of MFIs and MIIs. For this, we have defined two new global constraints, namely FREQUENTSUBS and INFREQUENTSUPERS, with polynomial complete propagators. We have proposed a new implementation of the infrequency constraint. We have proven that the problem of mining MFIs or MIIs with constraints is coNP-hard, ruling out the hope for a single CSP solving this problem (unless coNP $\subseteq$ NP). Nevertheless, our CP model is sound when the additional constraints are monotone on MFIs or anti-monotone on MIIs. Experiments showed that the CP approach is competitive with state of the art techniques for mining MFIs or MIIs and even better for mining MFIs or MIIs with additional constraints.

Conclusion

In this thesis, we have used constraint programming to model and solve some well known itemset mining problems. Our models offer the user the flexibility to express complex queries.

We first proposed a full constraint programming model for mining association rules. For this we have introduced a new global constraint CONFIDENT for the confidence of rules. We have proven that the problem of deciding if there exists a confident rule is NP-complete. This implies that domain consistency on CONFIDENT is NP-hard. We thus, proposed two weak propagators. The first one uses a filtering rule. The second one uses a decomposition using existing constraints. We have shown that, in addition to constrained rules, our model is able to capture minimal non-redundant rules. For this, we used a new global constraint, GENERATOR, for mining itemsets that are generators. We have proposed a complete polynomial propagator for GENERATOR. We also studied the problem of constrained minimal non-redundant rules. We have shown that anti-monotone constraints on the body and monotone constraints on the head are safe.

We have also proposed a generic constraint programming model for mining borders of frequent itemsets. For this, we have introduced two new global constraints, FREQUENTSUBS for itemsets that have only frequent subsets and INFREQUENTSUPERS for itemsets that have only infrequent supersets. For both constraints, we proposed polynomial complete propagators. We have proven that mining constrained borders is coNP-hard ruling out the possibility of a CSP solving this problem.

Experiments have shown the efficiency of our constraints and models. For mining classical patterns such as association rules, positive border, negative border, our CP approaches are not better than ad hoc methods. However, when it comes to mining patterns with additional constraints our CP approaches are flexible and faster thanks to constraint propagation. If we compare with declarative approaches, our CP approach for mining association rules outperforms the SAT approach. Despite the use of some constraint propagation to handle cardinality constraints, the SAT approach suffers from a memory issue when encoding the cover constraint into CNF.

We believe that constraint programming is a powerful paradigm that can be used to solve complex data mining problems. In the following we summarise some perspectives related to our work.

Short term perspectives

Extend/adapt our model for mining association rules

Association rules mining aims at detecting interesting correlations between items. Measuring this correlation remains a bottleneck nowadays. In some real world problems, the frequency and the confidence cannot fulfill all the user needs. Our constraint programming model for mining association rules can be extended to consider more complex statistical measures that describe the user's request best. Our CP model for mining association rules can be extended/adapted to mine other type of rules. Rare association rules, for instance, can be mined by just replacing the frequency constraint with the infrequency constraint in our model for mining association rules.

A global constraint with the confidence as variable

Our global constraint CONFIDENT can be revised to consider the confidence as a variable. In this case, the constraint CONFIDENTVAR$_{\mathcal{D}}(x, y, var)$ is satisfied if and only if $var = conf(x^{-1}(1) \rightarrow y^{-1}(1))$. The benefit of such constraint is the possibility to easily express constraints on the confidence (e.g., mine rules with a confidence less than a given threshold).

Long term perspectives

We can distinguish three main points that can be tackled as future works.

Constraint programming for high utility itemset mining

The frequency being monotonic offers the flexibility in designing algorithms for mining frequent itemsets. This is not true on the utility which is not monotonic. In itemset mining, high utility itemset mining consists in finding itemsets according to some given utilities (e.g., item prices). The itemset is considered to be of high utility if its utility is no less than a given threshold. Recently, the problem of high utility itemset mining is in the spotlight. Because the utility is not monotonic, it makes if difficult to design an efficient algorithm to handle the utility. In constraint programming the utility constraint can be

handled through a global constraint. A study of the level of consistency for the utility constraint is required to design a dedicated propagator. Existing CP models in itemset mining can then be extended to consider the utility constraint. Our model for mining association rules, for instance, can consider the utility constraint to mine rules that are of high utility in addition of being frequent and confident.

Preferences modeling

The user may ask for patterns with some preferences. These preferences can be quantitative according to some measure (e.g., a pattern is preferred to another according to its frequency) or qualitative according to a pairwise comparison between patterns (e.g., the pattern P_1 is preferred to P_2). Some of these preferences are not strong and can be violated. For instance, the user may prefer itemsets containing the item A unless all those itemsets are infrequent. These kind of preferences can be handled in constraint programming using soft constraints (i.e., weighted constraints) (Meseguer et al. [2006]). Weighted Constraint Satisfaction Problem (WCSP) is a CSP where some of the constraints can be violated according to some degrees (Bistarelli et al. [1999], Larrosa and Schiex [2004]). A generic CP model can be designed for itemset mining to take in consideration the soft constraints of the user in addition to the strong constraints.

Active mining using constraint acquisition

Active mining is the problem of mining patterns when we allow an interaction with the user (van Leeuwen [2014]). This problem can be divided into three main steps: (1) **mine:** where the computer extracts interesting patterns according to current information, (2) **interact:** where the computer interacts with the user who expresses her opinion on the extracted patterns, (3) **learn:** where the computer learns the user's constraints/preferences from the interaction. The three steps are repeated until the user is satisfied. In constraint programming the mining step can be handled by constraint solvers. The interaction and learning steps, on the other hand, are handled by constraint acquisition (Bessiere et al. [2017]). Constraint acquisition helps the user in the modeling process. Constraint acquisition interacts with the user via queries. Depending on the user's answer it learns (i.e., acquires) the constraints the user is interested to satisfy. The process is repeated until the user is satisfied. An architecture combining constraint solvers with constraint acquisition can be designed to actively mine patterns the user is interested in.

Bibliography

Mehdi Adda, Lei Wu, and Yi Feng. Rare itemset mining. In *The Sixth International Conference on Machine Learning and Applications, ICMLA 2007, Cincinnati, Ohio, USA, 13-15 December 2007* Adda et al. [2007], pages 73–80. doi: 10.1109/ICMLA.2007.106. URL https://doi.org/10.1109/ICMLA.2007.106. 5, v

Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan* Agrawal and Srikant [1995], pages 3–14. doi: 10.1109/ICDE.1995.380415. URL https://doi.org/10.1109/ICDE.1995.380415. 20, v

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993.* Agrawal et al. [1993], pages 207–216. doi: 10.1145/170035.170072. URL https://doi.org/10.1145/170035.170072. ix, 1, 3, 10, v

John O. R. Aoga, Tias Guns, and Pierre Schaus. Mining time-constrained sequential patterns with constraint programming. In *Constraints* Aoga et al. [2017], pages 548–570. doi: 10.1007/s10601-017-9272-3. URL https://doi.org/10.1007/s10601-017-9272-3. 31, v

Krzysztof R. Apt. *Principles of constraint programming*. In Apt [2003], 2003. ISBN 978-0-521-82583-2. 28, v

Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. In *Constraints* Asín et al. [2011], pages 195–221. doi: 10.1007/s10601-010-9105-0. URL https://doi.org/10.1007/s10601-010-9105-0. 37, v

Ahmed Ayad, Nagwa M. El-Makky, and Yousry I. Taha. Incremental mining of constrained association rules. In *Proceedings of the First SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, April 5-7, 2001* Ayad et al. [2001], pages 1–18. doi: 10.1137/1.9781611972719.24. URL https://doi.org/10.1137/1.9781611972719.24. 6, v

Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings* Bastide et al. [2000a], pages 972–986. doi: 10.1007/3-540-44957-4_65. URL https://doi.org/10.1007/3-540-44957-4_65. 13, v

Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. In *SIGKDD Explorations* Bastide et al. [2000b], pages 66–75. doi: 10.1145/380995.381017. URL https://doi.org/10.1145/380995.381017. 9, 15, vi

Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. In *J. ACM* Beeri et al. [1983], pages 479–513. doi: 10.1145/2402.322389. URL https://doi.org/10.1145/2402.322389. 27, vi

Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar. A global constraint for mining generator itemsets. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming, CP 2018, Lille, France, August 27-31, 2018* Belaid et al. [2018], page 11. URL http://cp2018.a4cp.org/dp-proceedings.pdf#page=92. xii, vi

Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar. Constraint programming for mining borders of frequent itemsets. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019* Belaid et al. [2019a], pages 1064–1070. doi: 10.24963/ijcai.2019/149. URL https://doi.org/10.24963/ijcai.2019/149. xii, vi

Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar. Constraint programming for association rules. In *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, Calgary, Alberta, Canada, May 2-4, 2019.* Belaid et al. [2019b], pages 127–135. doi: 10.1137/1.9781611975673.15. URL https://doi.org/10.1137/1.9781611975673.15. xii, 77, vi

Claude Berge. *Hypergraphs - combinatorics of finite sets*. Volume 45 of *North-Holland mathematical library* Berge [1989], 1989. ISBN 978-0-444-87489-4. 9, vi

Christian Bessiere. Arc-consistency and arc-consistency again. In *Artif. Intell.* Bessiere [1994], pages 179–190. doi: 10.1016/0004-3702(94)90041-8. URL https://doi.org/10.1016/0004-3702(94)90041-8. 27, vi

Christian Bessiere and Jean-Charles Régin. Refining the basic constraint propagation algorithm. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001* Bessiere and Régin [2001], pages 309–315. 27, vi

Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of reasoning with global constraints. In *Constraints* Bessiere et al. [2007], pages 239–259. doi: 10.1007/s10601-006-9007-3. URL https://doi.org/10.1007/s10601-006-9007-3. 45, vi

Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. In *Artif. Intell.* Bessiere et al. [2017], pages 315–342. doi: 10.1016/j.artint.2015.08.001. URL https://doi.org/10.1016/j.artint.2015.08.001. iii, vi

Christian Bessiere, Nadjib Lazaar, and Mehdi Maamar. User's constraints in itemset mining. In *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings* Bessiere et al. [2018], pages

537–553. doi: 10.1007/978-3-319-98334-9_35. URL https://doi.org/10.1007/978-3-319-98334-9_35. x, 31, vi

Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélène Fargier. Semiring-based csps and valued csps: Frameworks, properties, and comparison. In *Constraints* Bistarelli et al. [1999], pages 199–240. doi: 10.1023/A:1026441215081. URL https://doi.org/10.1023/A:1026441215081. iii, vii

Francesco Bonchi and Claudio Lucchese. On closed constrained frequent pattern mining. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK* Bonchi and Lucchese [2004], pages 35–42. doi: 10.1109/ICDM.2004.10093. URL https://doi.org/10.1109/ICDM.2004.10093. x, 18, 67, 75, vii

Francesco Bonchi and Claudio Lucchese. Pushing tougher constraints in frequent pattern mining. In *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings* Bonchi and Lucchese [2005], pages 114–124. doi: 10.1007/11430919_15. URL https://doi.org/10.1007/11430919_15. 17, vii

Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. On maximal frequent and minimal infrequent sets in binary matrices. In *Ann. Math. Artif. Intell.* Boros et al. [2003], pages 211–221. doi: 10.1023/A:1024605820527. URL https://doi.org/10.1023/A:1024605820527. 9, vii

Abdelhamid Boudane, Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. A sat-based approach for mining association rules. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016* Boudane et al. [2016], pages 2472–2478. URL http://www.ijcai.org/Abstract/16/352. x, 31, 36, 37, 56, vii

Abdelhamid Boudane, Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. Enumerating non-redundant association rules using satisfiability. In *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part I* Boudane et al. [2017], pages 824–836. doi: 10.1007/978-3-319-57454-7_64. URL https://doi.org/10.1007/978-3-319-57454-7_64. 36, 37, 39, 58, vii

Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany* Burdick et al. [2001], pages 443–452. doi: 10.1109/ICDE.2001.914857. URL https://doi.org/10.1109/ICDE.2001.914857. 7, vii

Maxime Chabert and Christine Solnon. Constraint programming for multi-criteria conceptual clustering. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings* Chabert and Solnon [2017], pages 460–476. doi: 10.1007/978-3-319-66158-2_30. URL https://doi.org/10.1007/978-3-319-66158-2_30. 31, vii

Raymond Chan, Qiang Yang, and Yi-Dong Shen. Mining high utility itemsets. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA* Chan et al. [2003], pages 19–26. doi: 10.1109/ICDM.2003.1250893. URL https://doi.org/10.1109/ICDM.2003.1250893. ix, viii

David Wai-Lok Cheung, Jiawei Han, Vincent T. Y. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana, USA* Cheung et al. [1996], pages 106–114. doi: 10.1109/ICDE.1996.492094. URL https://doi.org/10.1109/ICDE.1996.492094. 6, viii

Chun Kit Chui, Ben Kao, and Edward Hung. Mining frequent itemsets from uncertain data. In *Advances in Knowledge Discovery and Data Mining, 11th Pacific-Asia Conference, PAKDD 2007, Nanjing, China, May 22-25, 2007, Proceedings* Chui et al. [2007], pages 47–58. doi: 10.1007/978-3-540-71701-0_8. URL https://doi.org/10.1007/978-3-540-71701-0_8. 20, viii

Emmanuel Coquery, Saïd Jabbour, Lakhdar Saïs, and Yakoub Salhi. A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012), System Demonstrations Track, Montpellier, France, August 27-31 , 2012* Coquery et al. [2012], pages 258–263. doi: 10.3233/978-1-61499-098-7-258. URL https://doi.org/10.3233/978-1-61499-098-7-258. 31, viii

Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. In *Artif. Intell.* Dao et al. [2017], pages 70–94. doi: 10.1016/j.artint.2015.05.006. URL https://doi.org/10.1016/j.artint.2015.05.006. 31, viii

Romuald Debruyne and Christian Bessiere. Domain filtering consistencies. In *J. Artif. Intell. Res.* Debruyne and Bessiere [2001], pages 205–230. doi: 10.1613/jair.834. URL https://doi.org/10.1613/jair.834. 27, viii

Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. In *Artif. Intell.* Dechter [1990], pages 273–312. doi: 10.1016/0004-3702(90)90046-3. URL https://doi.org/10.1016/0004-3702(90)90046-3. 25, viii

Imen Ouled Dlala, Saïd Jabbour, Badran Raddaoui, Lakhdar Sais, and Boutheina Ben Yaghlane. A sat-based approach for enumerating interesting patterns from uncertain data. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016* Dlala et al. [2016], pages 255–262. doi: 10.1109/ICTAI.2016.0047. URL https://doi.org/10.1109/ICTAI.2016.0047. 31, viii

Imen Ouled Dlala, Saïd Jabbour, Badran Raddaoui, and Lakhdar Sais. A parallel sat-based framework for closed frequent itemsets mining. In *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings* Dlala et al. [2018], pages 570–587. doi: 10.1007/978-3-319-98334-9_37. URL https://doi.org/10.1007/978-3-319-98334-9_37. 36, viii

Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. In *J. ACM* Golomb and Baumert [1965], pages 516–524. doi: 10.1145/321296.321300. URL https://doi.org/10.1145/321296.321300. 23, 25, ix

Karam Gouda and Mohammed Javeed Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA* Gouda and Zaki [2001], pages 163–170. doi: 10.1109/ICDM.2001.989514. URL https://doi.org/10.1109/ICDM.2001.989514. 7, ix

Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA* Grahne and Zhu [2003]. URL http://ceur-ws.org/Vol-90/grahne.pdf. 77, ix

Peter Grünwald. A tutorial introduction to the minimum description length principle. In *CoRR* Grünwald [2004]. URL http://arxiv.org/abs/math.ST/0406077. 18, ix

David J. Haglin and Anna M. Manning. On minimal infrequent itemset mining. In *Proceedings of the 2007 International Conference on Data Mining, DMIN, 2007, June 25-28, 2007, Las Vegas, Nevada, USA* Haglin and Manning [2007], pages 141–147. 9, ix

Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.* Han and Fu [1995], pages 420–431. URL http://www.vldb.org/conf/1995/P420.PDF. 5, ix

Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.* Han et al. [2000], pages 1–12. doi: 10.1145/342009.335372. URL https://doi.org/10.1145/342009.335372. ix, 4

Saïd Jabbour, Mehdi Khiari, Lakhdar Sais, Yakoub Salhi, and Karim Tabia. Symmetry-based pruning in itemset mining. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013* Jabbour et al. [2013a], pages 483–490. doi: 10.1109/ICTAI.2013.78. URL https://doi.org/10.1109/ICTAI.2013.78. 36, ix

Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. The top-k frequent closed itemset mining using top-k SAT problem. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III* Jabbour et al. [2013b], pages 403–418. doi: 10.1007/978-3-642-40994-3_26. URL https://doi.org/10.1007/978-3-642-40994-3_26. 36, ix

Saïd Jabbour, Lakhdar Saïs, and Yakoub Salhi. A pigeon-hole based encoding of cardinality constraints. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2014, Fort Lauderdale, FL, USA, January 6-8, 2014* Jabbour et al. [2014]. URL http://www.cs.uic.edu/pub/Isaim2014/WebPreferences/ISAIM2014_Jabbour_etal.pdf. 37, ix

Saïd Jabbour, Fatima Ezzahra Mana, Imen Ouled Dlala, Badran Raddaoui, and Lakhdar Sais. On maximal frequent itemsets mining with constraints. In *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings* Jabbour et al. [2018], pages 554–569. doi: 10.1007/978-3-319-98334-9_36. URL https://doi.org/10.1007/978-3-319-98334-9_36. 36, 37, x

Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.* Jr. [1998], pages 85–93. doi: 10.1145/276304.276313. URL https://doi.org/10.1145/276304.276313. 7, x

Amina Kemmar, Samir Loudni, Yahia Lebbah, Patrice Boizumault, and Thierry Charnois. PREFIX-PROJECTION global constraint for sequential pattern mining. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings* Kemmar et al. [2015], pages 226–243. doi: 10.1007/978-3-319-23219-5_17. URL https://doi.org/10.1007/978-3-319-23219-5_17. 31, x

Amina Kemmar, Yahia Lebbah, and Samir Loudni. Interval graph mining. In *IJDMMM* Kemmar et al. [2018], pages 1–22. doi: 10.1504/IJDMMM.2018.10010657. URL https://doi.org/10.1504/IJDMMM.2018.10010657. 31, x

Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux. Constraint programming for mining n-ary patterns. In *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings* Khiari et al. [2010], pages 552–567. doi: 10.1007/978-3-642-15396-9_44. URL https://doi.org/10.1007/978-3-642-15396-9_44. x, 31

R. Uday Kiran and P. Krishna Reddy. Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings* Kiran and Reddy [2011], pages 11–20. doi: 10.1145/1951365.1951370. URL https://doi.org/10.1145/1951365.1951370. 5, x

Yun Sing Koh and Nathan Rountree. Finding sporadic rules using apriori-inverse. In *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings* Koh and Rountree [2005], pages 97–106. doi: 10.1007/11430919_13. URL https://doi.org/10.1007/11430919_13. 5, x

Marzena Kryszkiewicz. Representative association rules. In *Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD-98, Melbourne, Australia, April 15-17, 1998, Proceedings* Kryszkiewicz [1998a], pages 198–209. doi: 10.1007/3-540-64383-4_17. URL https://doi.org/10.1007/3-540-64383-4_17. 12, x

Marzena Kryszkiewicz. Representative association rules and minimum condition maximum consequence association rules. In *Principles of Data Mining and Knowledge Discovery, Second European Symposium, PKDD '98, Nantes, France, September 23-26, 1998,*

Proceedings Kryszkiewicz [1998b], pages 361–369. doi: 10.1007/BFb0094839. URL https://doi.org/10.1007/BFb0094839. 12, xi

Chan Man Kuok, Ada Wai-Chee Fu, and Man Hon Wong. Mining fuzzy association rules in databases. In *SIGMOD Record* Kuok et al. [1998], pages 41–46. doi: 10.1145/273244.273257. URL https://doi.org/10.1145/273244.273257. 14, xi

Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. In *Artif. Intell.* Larrosa and Schiex [2004], pages 1–26. doi: 10.1016/j.artint.2004.05.004. URL https://doi.org/10.1016/j.artint.2004.05.004. iii, xi

Nadjib Lazaar, Yahia Lebbah, Samir Loudni, Mehdi Maamar, Valentin Lemière, Christian Bessiere, and Patrice Boizumault. A global constraint for closed frequent pattern mining. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings* Lazaar et al. [2016], pages 333–349. doi: 10.1007/978-3-319-44953-1_22. URL https://doi.org/10.1007/978-3-319-44953-1_22. x, 31, 33, 34, 36, 48, xi

Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998* Lee and Stolfo [1998]. URL https://www.usenix.org/conference/7th-usenix-security-symposium/data-mining-approaches-intrusion-detection. 5, xi

Olivier Lhomme. Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993* Lhomme [1993], pages 232–238. URL http://ijcai.org/Proceedings/93-1/Papers/033.pdf. 27, xi

Ning Li, Li Zeng, Qing He, and Zhongzhi Shi. Parallel implementation of apriori algorithm based on mapreduce. In *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD, 2012, Kyoto, Japan, August 8-10, 2012* Li et al. [2012], pages 236–241. doi: 10.1109/SNPD.2012.31. URL https://doi.org/10.1109/SNPD.2012.31. 3, xi

Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. Apriori-based frequent itemset mining algorithms on mapreduce. In *The 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC '12, Kuala Lumpur, Malaysia, February 20-22, 2012* Lin et al. [2012], pages 76:1–76:8. doi: 10.1145/2184751.2184842. URL https://doi.org/10.1145/2184751.2184842. 3, xi

Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999* Liu et al. [1999], pages 337–341. doi: 10.1145/312129.312274. URL https://doi.org/10.1145/312129.312274. 5, xi

Alan K. Mackworth. Consistency in networks of relations. In *Artif. Intell.* Mackworth [1977a], pages 99–118. doi: 10.1016/0004-3702(77)90007-8. URL https://doi.org/10.1016/0004-3702(77)90007-8. 27, xi

Alan K. Mackworth. On reading sketch maps. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977* Mackworth [1977b], pages 598–606. URL http://ijcai.org/Proceedings/77-2/Papers/006.pdf. 27, xii

Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. In *Data Min. Knowl. Discov.* Mannila and Toivonen [1997], pages 241–258. doi: 10.1023/A:1009796218281. URL https://doi.org/10.1023/A:1009796218281. 7, 8, 9, 17, xii

Gou Masuda, Norihiro Sakamoto, and Ryuichi Yamamoto. A framework for dynamic evidence based medicine using data mining. In *15th IEEE Symposium on Computer-Based Medical Systems (CBMS 2002), 4-7 June 2002, Maribor, Slovenia* Masuda et al. [2002], pages 117–122. doi: 10.1109/CBMS.2002.1011364. URL https://doi.org/10.1109/CBMS.2002.1011364. 5, xii

Pedro Meseguer, Francesca Rossi, and Thomas Schiex. Soft constraints. In *Handbook of Constraint Programming* Meseguer et al. [2006], pages 281–328. doi: 10.1016/S1574-6526(06)80013-1. URL https://doi.org/10.1016/S1574-6526(06)80013-1. iii, xii

Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. In *Artif. Intell.* Mohr and Henderson [1986], pages 225–233. doi: 10.1016/0004-3702(86)90083-4. URL https://doi.org/10.1016/0004-3702(86)90083-4. 27, xii

Amit Anil Nanavati, Krishna Prasad Chitrapura, Sachindra Joshi, and Raghu Krishnapuram. Mining generalised disjunctive association rules. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001* Nanavati et al. [2001], pages 482–489. doi: 10.1145/502585.502666. URL https://doi.org/10.1145/502585.502666. 14, xii

Benjamin Négrevergne, Anton Dries, Tias Guns, and Siegfried Nijssen. Dominance programming for itemset mining. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013* Négrevergne et al. [2013], pages 557–566. doi: 10.1109/ICDM.2013.92. URL https://doi.org/10.1109/ICDM.2013.92. 68, 75, xii

Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained association rules. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.* Ng et al. [1998], pages 13–24. doi: 10.1145/276304.276307. URL https://doi.org/10.1145/276304.276307. 17, xii

Lhouari Nourine and Jean-Marc Petit. Extending set-based dualization: Application to pattern mining. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012), System Demonstrations Track, Montpellier, France, August 27-31 , 2012* Nourine and Petit [2012], pages 630–635. doi: 10.3233/978-1-61499-098-7-630. URL https://doi.org/10.3233/978-1-61499-098-7-630. 9, xii

Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995.* Park et al. [1995], pages 175–186. doi: 10.1145/223784.223813. URL https://doi.org/10.1145/223784.223813. 3, xiii

Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.* Pasquier et al. [1999], pages 398–416. doi: 10.1007/3-540-49257-7_25. URL https://doi.org/10.1007/3-540-49257-7_25. ix, 7, 8, xiii

Ronaldo Cristiano Prati, Maria Carolina Monard, and ACPLF Carvalho. A method for refining knowledge rules using exceptions. In *IV Encontro Nacional de Inteligncia Artificial (in portuguese, submitted)* Prati et al. [2003]. URL http://conteudo.icmc.usp.br/pessoas/mcmonard/public/asaiR2003.pdf. 5, xiii

Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008* Raedt et al. [2008], pages 204–212. doi: 10.1145/1401890.1401919. URL https://doi.org/10.1145/1401890.1401919. x, 31, 32, 44, xiii

Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.* Régin [1994], pages 362–367. URL http://www.aaai.org/Library/AAAI/1994/aaai94-055.php. 28, xiii

Jorma Rissanen. Modeling by shortest data description. In *Automatica* Rissanen [1978], pages 465–471. doi: 10.1016/0005-1098(78)90005-5. URL https://doi.org/10.1016/0005-1098(78)90005-5. 18, xiii

Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Volume 2 of Rossi et al. [2006], 2006. ISBN 978-0-444-52726-4. URL http://www.sciencedirect.com/science/bookseries/15746526/2. 23, xiii

Pierre Schaus, John O. R. Aoga, and Tias Guns. Coversize: A global constraint for frequency-based itemset mining. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings* Schaus et al. [2017], pages 529–546. doi: 10.1007/978-3-319-66158-2_34. URL https://doi.org/10.1007/978-3-319-66158-2_34. x, 31, 34, 35, 36, 43, 44, 47, 48, 58, 77, xiii

Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA* Siebes et al. [2006], pages 395–406. doi: 10.1137/1.9781611972764.35. URL https://doi.org/10.1137/1.9781611972764.35. 18, xiii

Laszlo Szathmary, Amedeo Napoli, and Sergei O. Kuznetsov. ZART: A multifunctional itemset mining algorithm. In *Proceedings of the Fifth International Conference on Concept*

Lattices and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007 Szathmary et al. [2007a]. URL http://ceur-ws.org/Vol-331/Szathmary.pdf. ix, xvii, 9, 15, 16, xiv

Laszlo Szathmary, Amedeo Napoli, and Petko Valtchev. Towards rare itemset mining. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), October 29-31, 2007, Patras, Greece, Volume 1* Szathmary et al. [2007b], pages 305–312. doi: 10.1109/ICTAI.2007.30. URL https://doi.org/10.1109/ICTAI.2007.30. 9, xiv

Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, and Robert Godin. An efficient hybrid algorithm for mining frequent closures and generators. In *Proceedings of the Fifth International Conference on Concept Lattices and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007* Szathmary et al. [2007c], pages 47–58. URL https://hal.inria.fr/inria-00335247/document. 56, 58, xiv

Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, and Robert Godin. Efficient vertical mining of frequent closures and generators. In *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings* Szathmary et al. [2009], pages 393–404. doi: 10.1007/978-3-642-03915-7_34. URL https://doi.org/10.1007/978-3-642-03915-7_34. 9, 15, 53, xiv

Laszlo Szathmary, Petko Valtchev, and Amedeo Napoli. Generating rare association rules using the minimal rare itemsets family. In *Int. J. Software and Informatics* Szathmary et al. [2010], pages 219–238. URL http://www.ijsi.org/ch/reader/view_abstract.aspx?file_no=i56. 14, xiv

Laszlo Szathmary, Petko Valtchev, Amedeo Napoli, and Robert Godin. Efficient vertical mining of minimal rare itemsets. In *Proceedings of The Ninth International Conference on Concept Lattices and Their Applications, Fuengirola (Málaga), Spain, October 11-14, 2012* Szathmary et al. [2012], pages 269–280. URL http://ceur-ws.org/Vol-972/paper23.pdf. 9, 77, xiv

Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Indirect association: Mining higher order dependencies in data. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings* Tan et al. [2000], pages 632–637. doi: 10.1007/3-540-45372-5_77. URL https://doi.org/10.1007/3-540-45372-5_77. 15, xiv

Luigi Troiano, Giacomo Scibelli, and Cosimo Birtolo. A fast algorithm for mining rare itemsets. In *Ninth International Conference on Intelligent Systems Design and Applications, ISDA 2009, Pisa, Italy , November 30-December 2, 2009* Troiano et al. [2009], pages 1149–1155. doi: 10.1109/ISDA.2009.55. URL https://doi.org/10.1109/ISDA.2009.55. 5, 6, xiv

Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA* Uno et al. [2003]. URL http://ceur-ws.org/Vol-90/uno.pdf. 8, xiv

Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004* Uno et al. [2004]. URL http://ceur-ws.org/Vol-126/uno.pdf. 7, 8, xv

Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations* Uno et al. [2005], pages 77–86. 8, xv

Matthijs van Leeuwen. Interactive data exploration using pattern mining. In *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics - State-of-the-Art and Future Challenges* van Leeuwen [2014], pages 169–182. doi: 10.1007/978-3-662-43968-5_9. URL https://doi.org/10.1007/978-3-662-43968-5_9. iii, xv

Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. In *Data Min. Knowl. Discov.* Vreeken et al. [2011], pages 169–214. doi: 10.1007/s10618-010-0202-x. URL https://doi.org/10.1007/s10618-010-0202-x. 18, xv

Jianyong Wang, Jiawei Han, and Jian Pei. CLOSET+: searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003* Wang et al. [2003], pages 236–245. doi: 10.1145/956750.956779. URL https://doi.org/10.1145/956750.956779. 8, xv

Ke Wang, Yu He, and Jiawei Han. Mining frequent itemsets using support constraints. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt* Wang et al. [2000a], pages 43–52. URL http://www.vldb.org/conf/2000/P043.pdf. 5, xv

Wei Wang, Jiong Yang, and Philip S. Yu. Efficient mining of weighted association rules (WAR). In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000* Wang et al. [2000b], pages 270–274. doi: 10.1145/347090.347149. URL https://doi.org/10.1145/347090.347149. 14, xv

Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. In *Inf. Process. Lett.* Warners [1998], pages 63–69. doi: 10.1016/S0020-0190(98)00144-6. URL https://doi.org/10.1016/S0020-0190(98)00144-6. 37, xv

Marek Wojciechowski and Maciej Zakrzewicz. Dataset filtering techniques in constraint-based frequent pattern mining. In *Pattern Detection and Discovery, ESF Exploratory Workshop, London, UK, September 16-19, 2002, Proceedings* Wojciechowski and Zakrzewicz [2002], pages 77–91. doi: 10.1007/3-540-45728-3_7. URL https://doi.org/10.1007/3-540-45728-3_7. ix, 50, xv

Xindong Wu, Chengqi Zhang, and Shichao Zhang. Efficient mining of both positive and negative association rules. In *ACM Trans. Inf. Syst.* Wu et al. [2004], pages 381–405. doi:

10.1145/1010614.1010616. URL https://doi.org/10.1145/1010614.1010616. 14, xv

Yanbin Ye and Chia-Chu Chiang. A parallel apriori algorithm for frequent itemsets mining. In *Fourth International Conference on Software Engineering, Research, Management and Applications (SERA 2006), 9-11 August 2006, Seattle, Washington, USA* Ye and Chiang [2006], pages 87–94. doi: 10.1109/SERA.2006.6. URL https://doi.org/10.1109/SERA.2006.6. 3, xvi

Mohammed Javeed Zaki. Scalable algorithms for association mining. In *IEEE Trans. Knowl. Data Eng.* Zaki [2000], pages 372–390. doi: 10.1109/69.846291. URL https://doi.org/10.1109/69.846291. 16, xvi

Mohammed Javeed Zaki. Mining non-redundant association rules. In *Data Min. Knowl. Discov.* Zaki [2004], pages 223–248. doi: 10.1023/B:DAMI.0000040429.96086.c7. URL https://doi.org/10.1023/B:DAMI.0000040429.96086.c7. 13, xvi