Mining-Based Compression Approach of Propositional Formulae

Said Jabbour, Lakhdar Sais, Yakoub Salhi
CRIL - CNRS, University of Artois
F-62307 Lens Cedex, France
{jabbour, sais, salhi}@cril.fr

Takeaki Uno
National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku
Tokyo 101-8430, Japan
uno@nii.jp

ABSTRACT

In this paper, we propose a first application of data mining techniques to propositional satisfiability. Our proposed mining based compression approach aims to discover and to exploit hidden structural knowledge for reducing the size of propositional formulae in conjunctive normal form (CNF). It combines both frequent itemset mining techniques and Tseitin's encoding for a compact representation of CNF formulae. The experimental evaluation of our approach shows interesting reductions of the sizes of many application instances taken from the last SAT competitions.

Categories and Subject Descriptors

F.4.1 [**Mathematical logic and formal languages**]: Mathematical Logic—*Logic and constraint programming*; H.2.8 [**Database management**]: Database applications—*Data mining*

Keywords

Compression; Data mining; Propositional satisfiability and modeling

1. INTRODUCTION

Propositional satisfiability (SAT) i.e., the problem of checking whether a Boolean formula in conjunctive normal form (CNF) is satisfiable or not, became a core technology in many application domains, such as formal verification, planning and various new applications derived by the recent impressive progress in practical SAT solving. SAT has gained a considerable audience with the advent of a new generation of solvers able to solve SAT instances with millions of variables and clauses [9, 4]. Today, these solvers represent an important low-level building block for many important fields, e.g., SAT modulo theory, Theorem proving, Model checking, Quantified Boolean formulas, Maximum Satisfiability, Pseudo Boolean, etc. In addition to the traditional applications of SAT to hardware and software formal verification, this impressive progress led to increasing use of SAT technology to solve new real-world applications such as in bioinformatics, cryptography and data mining.

Propositional formulae in CNF is the standard input format for propositional satisfiability. Indeed, most of the state-of-the-art SAT solvers are based on this normal form. Such convenient CNF representation is derived from a general Boolean formula using the well-known Tseitin encoding [14]. In practice, the general Boolean formula itself is usually derived from high level language in which problem structure is explicit.

Two important flaws were identified on the CNF form and largely discussed in the literature (e.g. [6]). First, it is often argued that by encoding arbitrary propositional formulae in CNF, structural properties of the original problem are not reflected in the CNF formula. Secondly, even if such translation is linear in the size of the original formula, a huge CNF formula might result when encoding real-world problems. Some instances exceed the capacity of the available memory, and even if the instance can be stored, the time needed for reading the input instance might be higher than its solving time. The obstacle here is not the potential difficulty of the instance, but its size. The growing success obtained in solving real-world SAT problems highlights a real transition to industrial and commercial scale. This results in a rapid growth in the size of the CNF instances encoding real-world problems. Consequently, the design of new efficient models for representing and for solving SAT instances of very large sizes (”Big” instances) is clearly an important challenge.

To address this problem, developing a more compact representation of CNF formulae is an interesting research issue. By compact encoding of formulae, we have in mind a representation model which through its use of structural knowledge results in the most compact possible formula equivalent with respect to satisfiability. By structural knowledge, we mean patterns that can be recognized or discovered. To be useful, such algorithm must be incremental.

Two promising models were proposed these last years. The first, proposed by H. Dixon et al [3], uses group theory to represent several classical clauses by a single clause called an "augmented clause". The second model was proposed by M. L. Ginsberg et al [5], called QPROP ("quantified propositional logic"), which may be seen as a propositional formula extended by the introduction of quantifications over finite domains, i.e. first order logic limited to finite types and without functional symbols. The problem rises in finding efficient solving techniques of formulae encoded using such models.

More recently, an original approach for compacting sets of binary clauses was proposed by J. Rintanen in [12]. Binary clauses are ubiquitous in propositional formulae that represent real-world problems ranging from model-checking problems in computer-aided verification to AI planning problems. In [12], using auxiliary variables, it is shown how constraint graphs that contain big cliques or bi-cliques of binary clauses can be represented more compactly than the quadratic and explicit representation. The main limitation of this approach lies in its restriction to particular sets of binary clauses whose constraints graph represents cliques or bi-cliques. Such particular regularities can be caused by the presence of an at-most-one constraint over a subset of Boolean variables, forbidding more than one of them to be true at a time.

In the data mining community, several models and techniques for discovering interesting patterns in large databases has been proposed in the last few years. The problem of mining frequent itemsets is well-known and essential in data mining, knowledge discovery and data analysis. Since the first article of Agrawal [1] on association rules and itemset mining, the huge number of works, challenges, datasets and projects show the actual interest in this problem (see [13] for a recent survey).

Our goal in this work is to address the problem of finding a compact representation of CNF formulae. Our proposed mining based compression approach aims to *discover hidden structures* from arbitrary CNF formulae and to exploit them to reduce the overall size of the CNF formula while preserving satisfiability. It is the first application of data mining techniques to Boolean Satisfiability.

Recently, a first constraint programming (CP) based data mining framework was proposed by Luc De Raedt et al. in [11, 2]. This new framework offers a declarative and flexible representation model. It allows data mining problems to benefit from several generic and efficient CP solving techniques [8]. This first study leads to the first CP approach for itemset mining displaying nice declarative opportunities while opening interesting perspectives to cross fertilization between data-mining, constraint programming and propositional satisfiability.

In this paper, we are particularly interested in the other side of this innovative connection between these two research domains, namely how data-mining can be helpful for SAT. We present the first data-mining approach for Boolean Satisfiability. We show that itemset mining techniques are very suitable for discovering interesting patterns from CNF formulae. Such patterns are then used to rewrite the CNF formula more compactly. We also show how sets of binary clauses can be also compacted by our approach. We also prove that our approach can automatically achieve similar reductions as in [12], on bi-cliques and cliques of binary clauses. It is also important to note, that our proposed mining4SAT approach is incremental. Indeed, our method can be applied incrementally or in parallel on the subsets of any partition of the original CNF formula. This will be particularly helpful for huge CNF formula that can not be entirely stored in memory.

tid	itemset
1	$Joyce, Beckett, Proust$
2	$Faulkner, Hemingway, Melville$
3	$Joyce, Proust$
4	$Hemingway, Melville$
5	$Flaubert, Zola$
6	$Hemingway, Golding$

Table 1: An example of transactions database $\mathcal{D}$

2. FREQUENT ITEMSET MINING PROBLEM

2.1 Preliminary Notations and Definitions

Let $\mathcal{I}$ be a set of *items*. A set $I \subseteq \mathcal{I}$ is called an *itemset*. A *transaction* is a couple (tid, I) where tid is the *transaction identifier* and I is an itemset. A *transactions database* $\mathcal{D}$ is a finite set of transactions over $\mathcal{I}$ where for all two different transactions, they do not have the same transaction identifier. We say that a transaction (tid, I) *supports* an itemset J if $J \subseteq I$.

The *cover* of an itemset I in a transactions database $\mathcal{D}$ is the set of identifiers of transactions in $\mathcal{D}$ supporting I: $\mathcal{C}(I,\mathcal{D}) = \{tid \mid (tid, J) \in \mathcal{D} \texttt{ and } I \subseteq J\}$. The *support* of an itemset I in $\mathcal{D}$ is defined by: $\mathcal{S}(I,\mathcal{D}) = \mid \mathcal{C}(I,\mathcal{D}) \mid$. Moreover, the *frequency* of I in $\mathcal{D}$ is defined by: $\mathcal{F}(I,\mathcal{D}) = \frac{\mathcal{S}(I,\mathcal{D})}{|\mathcal{D}|}$.

For example, let us consider the transactions database in Table 1. Each transaction corresponds to the favorite writers of a library member. For instance, we have :
$\mathcal{S}(\{Hemingway, Melville\},\mathcal{D}) = |\{2,4\}| = 2$ and
$\mathcal{F}(\{Hemingway, Melville\},\mathcal{D}) = \frac{1}{3}$.
Let $\mathcal{D}$ be a transactions database over $\mathcal{I}$ and λ a minimal support threshold. The frequent itemset mining problem consists of computing the following set: $\mathcal{FIM}(\mathcal{D},\lambda) = \{I \subseteq \mathcal{I} \mid \mathcal{S}(I,\mathcal{D}) \geqslant \lambda\}$.

The problem of computing the number of frequent itemsets is $\#P$-hard [7]. The complexity class $\#P$ corresponds to the set of counting problems associated with a decision problem in NP. For example, counting the number of models satisfying a CNF formula is a $\#P$ problem.

2.2 Maximal and Closed Frequent Itemsets

Let us now recall two popular condensed representations of the set of all frequent itemsets: maximal and closed frequent itemsets.

Definition 1 (Maximal Frequent Itemset). *Let $\mathcal{D}$ be a transactions database, λ a minimal support threshold and $I \in \mathcal{FIM}(\mathcal{D},\lambda)$. I is called maximal when for all $I' \supset I$, $I' \notin \mathcal{FIM}(\mathcal{D},\lambda)$ (I' is not a frequent itemset).*

We denote by $\mathcal{MAX}(\mathcal{D},\lambda)$ the set of all maximal frequent itemsets in $\mathcal{D}$ with λ as a minimal support threshold. For instance, in the previous example, we have $\mathcal{MAX}(\mathcal{D},2) = \{\{Joyce, Proust\},\{Hemingway, Melville\}\}$.

Definition 2 (Closed Frequent Itemset). *Let $\mathcal{D}$ be a transactions database, λ a minimal support threshold and $I \in \mathcal{FIM}(\mathcal{D}, \lambda)$. I is called closed when for all $I' \supset I$, $\mathcal{C}(I, \mathcal{D}) \neq \mathcal{C}(I', \mathcal{D})$.*

We denote by $\mathcal{CLO}(\mathcal{D}, \lambda)$ the set of all closed frequent itemsets in $\mathcal{D}$ with λ as a minimal support threshold. For instance, we have $\mathcal{CLO}(\mathcal{D}, 2) = \{\{Hemingway\}, \{Joyce, Proust\}, \{Hemingway, Melville\}\}$. In particular, let us note that we have $\mathcal{C}(\{Hemingway\}, \mathcal{D}) = \{2, 4, 6\}$ and $\mathcal{C}(\{Hemingway, Melville\}, \mathcal{D}) = \{2, 4\}$. That explains why $\{Hemingway\}$ and $\{Hemingway, Melville\}$ are both closed. One can easily see that if all the closed (resp. maximal) frequent itemsets are computed, then all the frequent itemsets can be computed without using the corresponding transactions database. Indeed, the frequent itemsets correspond to all the subsets of the closed (resp. maximal) frequent itemsets.

The number of maximal (resp. closed) frequent itemsets is significantly smaller than the number of frequent itemsets. Nonetheless, this number is not always polynomial in the size of the database [16]. In particular, the problem of counting the number of maximal frequent itemsets is $\#P$-complete (see also [16]).

Many algorithm has been proposed for enumerating frequent closed itemsets. One can cite Apriori-like algorithm, originally proposed in [1] for mining frequent itemsets for association rules. It proceeds by a level-wise search of the elements of $\mathcal{FIM}(\mathcal{D}, \lambda)$. Indeed, it starts by computing the elements of $\mathcal{FIM}(\mathcal{D}, \lambda)$ of size one. Then, assuming the element of $\mathcal{FIM}(\mathcal{D}, \lambda)$ of size n is known, it computes a set of candidates of size $n + 1$ so that I is a candidate if and only if all its subsets are in $\mathcal{FIM}(\mathcal{D}, \lambda)$. This procedure is iterated until no more candidates are found. Obviously, this basic procedure is enhanced using some properties such as the anti-monotonicity property that allow us to reduce the search space. Indeed, if $I \notin \mathcal{FIM}(\mathcal{D}, \lambda)$, then $I' \notin \mathcal{FIM}(\mathcal{D}, \lambda)$ for all $I' \supseteq I$. In our experiments, we consider one of the state-of-the-art algorithms LCM for mining frequent closed itemsets proposed by Takeaki Uno et al. in [15]. In theory, the authors prove that LCM exactly enumerates the set of frequent closed itemsets within polynomial time per closed itemset in the total input size. Let us mention that LCM algorithm obtained the best implementation award of FIMI'2004 (Frequent Itemset Mining Implementations).

3. FROM CNF FORMULA TO TRANSACTION DATABASE

We first introduce the satisfiability problem and some necessary notations. We consider the conjunctive normal form (CNF) representation for the propositional formulas. A *CNF formula* Φ is a conjunction of clauses, where a *clause* is a disjunction of literals. A *literal* is a positive (p) or negated ($\neg p$) propositional variable. The two literals p and $\neg p$ are called *complementary.* A CNF formula can also be seen as a set of clauses, and a clause as a set of literals. The size of the CNF formula Φ is defined as $|\Phi| = \sum_{c \in \Phi} |c|$, where $|c|$ is equal to the number of literals in c. A *unit* clause is a clause containing only one literal (called *unit literal*), while a binary clause contains exactly two literals. A formula containing only binary clauses is called *2-CNF* formula. An *empty clause*, denoted $\perp$, is interpreted as false (unsatisfiable), whereas an *empty CNF formula*, denoted $\top$, is interpreted as true (satisfiable).

Let c_1 and c_2 be two clauses of a formula Φ. We say that c_1 *subsumes* c_2 iff $c_1 \subseteq c_2$. If c_1 subsumes c_2, then the clause c_2 can be deleted from Φ while preserving satisfiability. A CNF formula Φ is *closed under subsumption* iff $\forall c \in \Phi$, $\nexists c' \in \Phi$ such that $c \neq c'$ and c' subsumes c. We denote by Φ^s, the formula obtained from Φ by eliminating all the subsumed clauses.

We note $\bar{l}$ the *complementary* literal of l. More precisely, if $l = p$ then $\bar{l}$ is $\neg p$ and if $l = \neg p$ then $\bar{l}$ is p. Let us recall that any propositional formula can be translated to CNF using Tseitin's linear encoding [14]. We denote by $\mathcal{V}_\Phi$ the set of propositional variables appearing in Φ, while the set of literals of Φ is defined as $\mathcal{L}_\Phi$.

An *interpretation* $\mathcal{B}$ of a propositional formula Φ is a function which associates a value $\mathcal{B}(p) \in \{0, 1\}$ (0 corresponds to $false$ and 1 to *true*) to the variables $p \in \mathcal{V}_\Phi$. A *model* of a formula Φ is an interpretation $\mathcal{B}$ that satisfies the formula: $\mathcal{B}(\Phi) = 1$. The *SAT problem* consists in deciding if a given CNF formula admits a model or not.

We define $\Phi|_x$ as the formula obtained from Φ by assigning x the truth-value 1. Formally $\Phi|_x = \{c \mid c \in \Phi, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \Phi, \neg x \in c\}$. Φ^* denotes the formula Φ *closed under unit propagation*, defined recursively as follows: (1) $\Phi^* = \Phi$ if Φ does not contain any unit clause, (2) $\Phi^* = \perp$ if Φ contains two unit-clauses $\{x\}$ and $\{\neg x\}$, (3) otherwise, $\Phi^* = (\Phi|_x)^*$ where x is the literal appearing in a unit clause of Φ.

A CNF formula can be considered as a transactions database, called CNF database, where the items correspond to literals and the transactions to clauses. Note that complementary literals correspond to two different items.

Definition 3 (CNF to $\mathcal{D}$). *Let $\Phi = \bigwedge_{1 \leqslant i \leqslant n} c_i$ be a CNF formula. The set of items $\mathcal{I} = \mathcal{L}_\Phi$ and the transactions database associated to Φ is defined as $\mathcal{D}^c_\Phi = \{(tid_i, c_i) | 1 \leqslant i \leqslant n\}$*

For instance, the CNF formula $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge x_1 \wedge (x_3 \vee \neg x_4)$ corresponds to the following transactions database:

tid	itemset
1	$x_1, \neg x_2, \neg x_3$
2	$x_1, \neg x_2, x_4$
3	x_1
4	$x_3, \neg x_4$

In this context, a frequent itemset corresponds to a frequent set of literals: the number of clauses containing these literals is greater or equal to the minimal support threshold. For instance, if we set the minimal threshold λ to 2, we get $\{x_1, \neg x_2\}$ as a frequent itemset in the previous database. The set of maximal frequent itemsets is the smallest set of frequent set of literals where each frequent set of literals is included in at least one of its elements. For instance, the unique maximal frequent itemset in the previous example is $\{x_1, \neg x_2\}$ ($\lambda = 2$). Furthermore, the set of closed frequent itemsets is the smallest set of frequent set of literals where each frequent itemset is included in at least one of its elements *having the same support.* For instance, the set of the closed frequent itemsets is $\{\{x_1, \neg x_2\}, \{x_1\}\}$.

In the definition of a transaction database, we did not require that the set of items in a transaction to be unique. Indeed, two different transactions can have the same set of items and different identifiers. A CNF formula may contain the same clause more than once, but in practice this does not provide any information about satisfiability. Thus, we can consider a CNF database as just a set of itemsets (sets of literals).

4. MINING-BASED APPROACH FOR SIZE-REDUCTION OF CNF FORMULAE

In this section, we describe our mining based approach, called Mining4SAT, for reducing the size of CNF formulae. The key idea consists in searching for frequent sets of literals (sub-clauses) and substituting them with new variables using Tseitin's encoding [14].

4.1 Tseitin's Encoding

Tseitin's encoding consists in introducing fresh variables to represent sub-formulae in order to represent their truth values. For example, given a Boolean formula, containing the variables a and b, and v a fresh variable, one can add the definition $v \leftrightarrow a \vee b$ (called extension) to the formula while preserving satisfiability. Tseitin's extension principle is at the basis of the linear transformation of general Boolean formulae into CNF. Two decades later, after Tseitin's seminal paper, Plaisted and Greenbaum presented an improved CNF translation that essentially produces a subset of Tseitin's representation [10]. They noticed that by keeping track of polarities of sub-formulae, one can remove large parts of Tseitin translation. In the sequel, we use Plaisted and Greenbaum improvement. More precisely, as the disjunction $a \vee b$ is a sub-clause with positive polarity, it is sufficient to add the formula $v \rightarrow a \vee b$ i.e. a clause $(\neg v \vee a \vee b)$.

Let consider the following DNF formula (Disjunctive Normal Form: a disjunction of conjunctions):

$$(x_1 \wedge \cdots \wedge x_l) \vee (y_1 \wedge \cdots \wedge y_m) \vee (z_1 \wedge \cdots \wedge z_n)$$

A naive way of converting such a formula to a CNF formula consists in using the distributivity of disjunction over conjunction $(A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C))$:

$$(x_1 \vee y_1 \vee z_1) \wedge (x_1 \vee y_1 \vee z_2) \wedge \cdots \wedge (x_l \vee y_m \vee z_n)$$

Such a naive approach is clearly exponential in the worst case. In Tseitin's transformation, fresh propositional variables are introduced to prevent such combinatorial explosion, mainly caused by the distributivity of disjunction over conjunction and vice versa. With additional variables, the obtained CNF formula is linear in the size of the original formula. However the equivalence is only preserved w.r.t satisfiability:

$$(t_1 \vee t_2 \vee t_3) \wedge (t_1 \rightarrow (x_1 \wedge \cdots \wedge x_l)) \wedge (t_2 \rightarrow (y_1 \wedge \cdots \wedge y_m))$$
$$\wedge (t_3 \rightarrow (z_1 \wedge \cdots \wedge z_n))$$

4.2 A Size-Reduction Method

Let us now describe in more details, how itemset mining techniques can be combined with Tseitin's principle to compress CNF formula.

To illustrate the main ideas behind our mining based compression approach, we consider the CNF formula Φ:

$$(x_1 \vee \cdots \vee x_n \vee \alpha_1) \wedge \cdots \wedge (x_1 \vee \cdots \vee x_n \vee \alpha_k)$$

where $n \geqslant 2$, $k > \frac{n+1}{n-1}$ and $\alpha_1, \ldots, \alpha_k$ are clauses. As we can observe, the sub-clause $(x_1 \vee \ldots, \vee x_n)$ appears in each clause of Φ. Using Tseitin's encoding, we can rewrite Φ as follows:

$$(y \vee \alpha_1) \wedge \cdots \wedge (y \vee \alpha_k) \wedge (x_1 \vee \cdots \vee x_n \vee \neg y)$$

where y is a fresh propositional variable. Indeed, $n \times k$ literals are replaced with $k + n + 1$ literals leading to a gain in terms of number of literals of $(n \times k) - (n + k + 1)$.

Now, if we consider the CNF database corresponding to Φ, $\{x_1, \ldots, x_n\}$ is a frequent itemset where the minimal support threshold is greater or equal to k. It is easy to see that to reduce the number of literals n must be greater or equal to 2. Indeed, if $n < 2$ then there is no reduction of the number of literals, on the contrary, their number is increased. Regarding the value of k, one can also see that such a transformation is interesting only when $k > \frac{n+1}{n-1}$. Thus, there are three cases : if $n = 2$, then $k \geqslant 4$, else if $n = 3$ then $k \geqslant 3$, $k \geqslant 2$ otherwise. Therefore, the number of literals is always reduced when $k \geqslant 4$.

Obviously, a boolean interpretation is a model of the formula obtained after reduction if and only if it is a model of Φ.

In the previous example, we illustrate how the problem of finding frequent itemsets can be used to reduce the size of a CNF formula. One can see that, in general, it is more interesting to consider a condensed representation of the frequent itemsets (closed and maximal) to reduce the number of literals. Indeed, by using a condensed representation, we consider all the frequent itemsets and the number of fresh propositional variables and new clauses (in our example, y and $(x_1 \vee \cdots \vee x_n \vee \neg y)$) introduced is smaller than that of those introduced by using all the frequent itemsets. For instance, in the previous formula, it is not interesting to introduce a fresh propositional variable for each subset of $\{x_1, \ldots, x_n\}$.

EXAMPLE 1. *Let us consider a formula Φ containing the following 8 clauses:*

$$\begin{array}{lcl}
\neg x_0 \vee x_2 & & \\
\neg x_1 \vee x_4 & & \\
\neg x_3 \vee x_4 & & \\
\neg x_5 \vee x_6 & & \\
\hline
\neg x_0 \vee x_1 & \vee & |\, x_4 \vee x_5 \vee x_6 \,| \\
x_3 & \vee & |\, x_4 \vee x_5 \vee x_6 \,| \\
\neg x_1 \vee x_2 & \vee & |\, x_4 \vee x_5 \vee x_6 \,| \\
\neg x_2 \vee x_3 & \vee & |\, x_4 \vee x_5 \vee x_6 \,| \\
\hline
\end{array}$$

Suppose that the minimal support threshold is less than 4, then the sub-clause $(x_4 \vee x_5 \vee x_6)$ is frequent. Using our approach, the formula Φ can be rewritten as:

$$\begin{array}{lcl}
\neg x_0 \vee x_2 & & \\
\neg x_1 \vee x_4 & & \\
\neg x_3 \vee x_4 & & \\
\neg x_5 \vee x_6 & & \\
\hline
\neg x_0 \vee x_1 & \vee & |\, \mathbf{y} \,| \\
x_3 & \vee & |\, \mathbf{y} \,| \\
\neg x_1 \vee x_2 & \vee & |\, \mathbf{y} \,| \\
\neg x_2 \vee x_3 & \vee & |\, \mathbf{y} \,| \\
\hline
\neg \mathbf{y} & \vee & x_4 \vee x_5 \vee x_6
\end{array}$$

In this simple example, the original formula contains 27 *literals, while the new formula involves only* 23 *literals.*

Closed vs. Maximal.

In Section 2.2, we introduced two condensed representations of the frequent itemsets: closed and maximal. The question is, which condensed representation is better? We know that the set of maximal frequent itemsets is included in that of the closed ones. Thus, a small number of fresh variables and new clauses are introduced using the maximal frequent itemsets. However, there are cases where the use of the closed frequent itemsets is more suitable. For example, let us consider the following formula:

$$\begin{array}{ll} (x_1 \vee \ldots \vee x_k \vee \ldots \vee x_n \vee \alpha_1) & \wedge \\ \vdots & \vdots \\ (x_1 \vee \ldots \vee x_k \vee \ldots \vee x_n \vee \alpha_m) & \wedge \\ (x_1 \vee \ldots \vee x_k \vee \beta_1) & \wedge \\ \vdots & \vdots \\ (x_1 \vee \ldots \vee x_k \vee \beta_{m'}) & \end{array}$$

where $k \geqslant 2$, $m, m' \geqslant 4$ and $n > k$. We assume that the frequent itemsets are only the subsets of $\{x_1, \ldots, x_n\}$. Therefore, $\{x_1, \ldots, x_n\}$ is the unique maximal itemset and the closed itemsets are $\{x_1, \ldots, x_n\}$ and $\{x_1, \ldots, x_k\}$. Let us start by using the closed frequent itemset $\{x_1, \ldots, x_n\}$ in the reduction of the number of literals:

$$\begin{array}{ll} (y \vee \alpha_1) & \wedge \\ \vdots & \vdots \\ (y \vee \alpha_m) & \wedge \\ (x_1 \vee \ldots \vee x_k \vee \beta_1) & \wedge \\ \vdots & \vdots \\ (x_1 \vee \ldots \vee x_k \vee \beta_{m'}) & \wedge \\ (x_1 \vee \ldots \vee x_n \vee \neg y) & \end{array}$$

Now, by using $\{x_1, \ldots, x_k\}$, we get the following formula:

$$\begin{array}{ll} (y \vee \alpha_1) & \wedge \\ \vdots & \vdots \\ (y \vee \alpha_m) & \wedge \\ (z \vee \beta_1) & \wedge \\ \vdots & \vdots \\ (z \vee \beta_{m'}) & \wedge \\ (z \vee x_{k+1} \vee \ldots \vee x_n \vee \neg y) & \wedge \\ (x_1 \vee \ldots \vee x_k \vee \neg z) & \end{array}$$

In this example, it is more interesting to consider the closed frequent itemsets in our Mining4SAT approach.

In fact, a (closed) frequent itemset I and one of its subsets I' (which can be closed) are both interesting if $\mathcal{S}(I') - \mathcal{S}(I) > \frac{|I'|+1}{|I'|-1} - 1$. Indeed, if we apply our transformation using I, then the support of I' in the resulting formula is equal to $\mathcal{S}(I') - \mathcal{S}(I) + 1$, and we know that I' is interesting in the resulting formula if its support is greater to $\frac{|I'|+1}{|I'|-1}$.

Overlap.

Let Φ be a set of itemsets. Two itemsets I and I' of Φ overlap if $I \cap I' \neq \emptyset$. Moreover, I and I' are in the same *overlap class* if there exist k itemsets $I_1, \ldots, I_k$ of Φ such that $I = I_1, I_k = I'$ and for all $1 \leqslant i \leqslant k-1$, I_i and I_{i+1} overlap.

In our transformation, one can have some problems when two frequent itemsets overlap. For example, if $\{x_1, x_2, x_3\}$ and $\{x_2, x_3, x_4\}$ are two frequent itemsets (3 is the minimal support threshold) such that $\mathcal{S}(\{x_1, x_2, x_3\}) = 3$, $\mathcal{S}(\{x_2, x_3, x_4\}) = 3$ and $\mathcal{S}(\{x_1, x_2, x_3, x_4\}) = 2$, then if we apply our transformation using $\{x_1, x_2, x_3\}$, then the support of $\{x_2, x_3, x_4\}$ is equal to 2 (infrequent) in the resulting formula and vice versa. Thus, we can not use both of them in the transformation.

Le us note that the overlap notion can be seen as a generalization of the subset one. Let I and I' be frequent itemsets such that they overlap. They are both interesting in our transformation if:

1. $\mathcal{S}(I) - \mathcal{S}(I \cup I') > \frac{|I|+1}{|I|-1} - 1$ or $\mathcal{S}(I') - \mathcal{S}(I \cup I') > \frac{|I'|+1}{|I'|-1} - 1$. This comes from the fact that if we apply the transformation using I (resp. I'), then the support of I' (resp. I) is equal to $\mathcal{S}(I') - \mathcal{S}(I \cup I') + 1$ (resp. $\mathcal{S}(I) - \mathcal{S}(I \cup I') + 1$).

2. $|I \backslash I'| \geqslant k$ (resp. $|I' \backslash I| \geqslant k$) where $k = 2$ if $\mathcal{S}(I) \geqslant 4$ (resp. $\mathcal{S}(I') \geqslant 4$), $k = 3$ if $\mathcal{S}(I) = 3$ (resp. $\mathcal{S}(I') = 3$), $k = 4$ otherwise. Indeed, in the previous cases, $I \backslash I'$ (resp. $I' \backslash I$) can be used in our transformation.

Mining4SAT algorithm.

We now describe our compression algorithm, called Mining4SAT, using the set of closed frequent itemsets. Let us note that the optimal transformation using the set of all the closed frequent itemsets can be obtained by an optimal transformation using separately the overlap classes of this set. Actually, since any two distinct overlap classes do not share any literal, the reduction applied to a given formula using the elements of an overlap class does not affect the supports of the elements of the other classes. Moreover, one can easily compute the set of all the overlap classes of the set of the closed frequent itemsets: let $G = (V, E)$ be an undirected graph such that V is the set of the closed frequent itemsets and (I_1, I_2) is an edge of G if and only if I_1 and I_2 overlap; C is an overlap class if and only if it corresponds to the set of vertices of a connected component of G which is not included in any other connected component of G. For this reason, we restrict here our attention to the reductions that can be obtained using a single overlap class. The whole size reduction process can be performed by iterating on all the overlap classes.

Let I be a closed frequent itemset, we denote by $\alpha(I)$ the value $\mathcal{S}(I) \times (|I|-1) - |I| - 1$ that corresponds to the number of literals reduced by applying our transformation with I on a CNF formula.

Algorithm 1 takes as input a CNF formula ϕ and an overlap class C, and returns ϕ after applying size-reduction transformations. It iterates until there is no element in C. In each iteration, it first selects one of the most interesting elements in C (line 2): an element I of C such that there is no element $I' \in C$ satisfying $\alpha(I') > \alpha(I)$. Note that this element is not necessarily unique in C. This instruction means that Algorithm 1 is a greedy algorithm because it makes a locally optimal choice at each iteration. Then, it applies

Algorithm 1 Size Reduction

```
Require: A formula $\phi$, an overlap class of closed frequent itemsets $C$
 1: while $C \neq \emptyset$ do
 2:     $I \leftarrow MostInterstingElment(C)$;
 3:     $replace(\phi, I, x)$;
 4:     $Add(\phi, I, x)$:
 5:     $remove(C, I)$;
 6:     $replaceSubset(C, I, x)$;
 7:     $removeUninterestingElements(C)$;
 8:     $updateSupports(C)$;
 9: end while
10: return $\phi$
```

our transformation using $I = \{y_1, \dots, y_n\}$: it replaces the occurrences of I with a fresh propositional variable x (line 3); and it adds the clause $y_1 \vee \dots \vee y_n \vee \neg x$ to ϕ (line 4). It next removes I from C (line 5) and replaces I in the the other elements of C with x (line 6). The next instruction (line 7) consists in removing the elements of C that could increase the number of literals: the elements that overlap with I and are not included in I. As explained before, an element of C overlapping with I does not necessarily increase the number of literals. Thus, by removing elements from C because only they overlap with I, our algorithm can remove closed frequent itemsets decreasing the number of literals. A partial solution to this problem consists in recomputing the closed frequent itemsets in the formula returned by Algorithm 1. The last instruction in the while loop (line 8) consists in updating the supports of the elements remaining in C following the new value of ϕ: a support of an element I' remaining in C changes only when it is included in I and its new support is equal to $\mathcal{S}(I') - \mathcal{S}(I) + 1$. This instruction also removes all the elements of C becoming uninteresting because of the new supports and sizes.

5. APPLICATION: A COMPACT REPRESENTATION OF SETS OF BINARY CLAUSES

Binary clauses (2-CNF formula) are ubiquitous in CNF formula encoding real-world problems. Some of them contain more than 90% of binary clauses. One of the main reason is that the encoding of several kinds of constraints to CNF leads to big sets of binary clauses. As an example, expressing that the variables $x_1, \dots, x_n$ must take different values in $\{v_1, \dots, v_m\}$ can leads to $n^2 \times m^2$ binary clauses (cliques of binary clauses) with a naive encoding. For $n = 100$ and $m = 10$, we get about one million of binary clauses or 10 Megabytes if each binary clause takes 10 bytes. Another example given by Rintanen in [12], concerns the encoding of planning problem to SAT and particularly of some invariants such as the one expressing that an object cannot be in two locations at the same time. For n state variables there are $\frac{n \times (n-1)}{2}$ invariants that are binary clauses. In the case of a planning problem with $n = 5000$ state variables and a formula that encodes the search for plans of 100 time points, if only one of the state variables is true at any given time, the total size of the set of binary clauses is about 12 Gigabytes.

Table 2 illustrates the proportion of binary clauses on a sample of SAT instances (application category) taken from the last SAT competitions [17]. For each instance (first column), we give the total number of clauses (#cls), the number of binary clauses (#bin) and the ratio of binary clauses ((%)bin).

instance	#cls	#bin	(%) bin
velev-pipe-o-uns-1.1-6	304026	268354	88,26 %
9dlx_vliw_at_b_iq2	542253	500227	92,24 %
1dlx_c_iq57_a	8562505	7567948	88,38 %
7pipe_k	751116	722278	96,16 %
SAT_dat.k100.debugged	670701	523153	78,00 %
BM_FV_2004_rule_batch	445444	339588	76,23 %
sokoban-sequential-p145-*.040-*	1413816	1364160	96,48 %
openstacks-*-p30_1.085-*	1621926	1601145	98,71 %
aaai10-planning-ipc5-*-12-step16	1029036	991140	96,31 %
k2fix_gr_rcs_w8.shuffled	271393	270136	99,53 %
homer17.shuffled	1742	1716	98,50 %
gripper13u.shuffled-as.sat03-395	38965	35984	92,34 %
grid-strips-grid-y-3.045-*	2750755	2695230	97,98 %

Table 2: Ratio of binary clauses in some SAT instances

In our Mining4SAT general approach presented previously, binary clauses are not taken into account. Indeed, to reduce the size of a formula, we only search for itemsets of size at least two. The case where a binary clause representing a closed frequent itemset can be considered by our Mining4SAT algorithm is when it appears at least four times in a formula Φ. For example, when Φ contains the following clauses $c_1 = (a \vee b)$, $c_2 = (a \vee b \vee \alpha_1)$, $c_3 = (a \vee b \vee \alpha_2)$, and $c_4 = (a \vee b \vee \alpha_3)$, where α_i for $1 \leqslant i \leqslant 3$ are clauses. In this case, the last three clauses are subsumed by the first binary clause. The clauses c_2, c_3 and c_4 are redundant and can be eliminated from Φ. Consequently, if we suppose that a formula is closed under subsumption, this case do not happen. Let us give a more general formulation of this particular case. Let Φ be the following formula:

$$\begin{array}{ll}(x_1 \vee \cdots \vee x_n) & \wedge \\ (x_1 \vee \cdots \vee x_n \vee \alpha_1) & \wedge \\ \vdots & \vdots \\ (x_1 \vee \cdots \vee x_n \vee \alpha_k) & \end{array}$$

where $\alpha_1, \dots, \alpha_k$ are clauses. As we can see, the first clause subsumes all the remaining clauses. Then we obtain the formula closed under subsumption $\Phi^s = (x_1 \vee \cdots \vee x_n)$. Suppose that $I = \{x_1, \cdots, x_n\}$ is a frequent closed itemset of $\mathcal{D}^c_\Phi$. Using I, we obtain the following formula Φ':

$$(y) \wedge (y \vee \alpha_1) \wedge \cdots \wedge (y \vee \alpha_k) \wedge (\neg y \vee x_1 \vee \cdots \vee x_n)$$

Applying unit propagation, we obtain $\Phi'^* = (x_1 \vee \cdots \vee x_n)$. As we can remark $\Phi^s = \Phi'^*$. To summarize, our general Mining4SAT can derive unit literals (y). Then by applying unit propagation closure, some redundant clauses are eliminated automatically.

5.1 Compacting Arbitrary Sets of Binary Clauses

In this section, we first show how our mining based approach can be used to achieve a compact representation of arbitrary set of binary clauses. Then, we consider two interesting special cases corresponding to sets of binary clauses representing either a clique or a bi-clique. It is important to note that, in [12], the authors investigated only these particular cases.

In order to make strong reductions in terms of literals but also in terms of clauses, we propose a four step approach for the compression of a set of binary clauses. In the first step, we rewrite the set of binary clauses using another more suitable representation. Secondly, from this intermediary representation, we derive a new transactions database. Then we search for frequent closed itemsets. Finally, we apply the compression algorithm obtained by a slight modification to the compression Algorithm 1 on the set of binary clauses .

Let us first introduce a more convenient and equivalent representation of a set binary of clauses.

Definition 4 (B-implication). *A B-implication is a Boolean formula of the following from : $x \vee \beta(x)$ where $\beta(x)$ is a conjunction of literals.*

Let S be the following set of binary clauses: $\{(x \vee y_1), \dots, (x \vee y_k)\}$, with $k \geqslant 1$. Note that the B-implication $B(S) = x \vee (y_1 \wedge \dots \wedge y_k)$ is equivalent to the conjunction of the elements of S. Hence, each 2-CNF formula Φ can be transformed into a set of B-implications, noted $B_{[\vee(\wedge)]}(\Phi)$. The original formula Φ can be obtained from $B_{[\vee(\wedge)]}(\Phi)$ by distributing $\vee$ over $\wedge$. However, there exist several ways for rewriting a 2-CNF as a conjunction of B-implications. A naive way is to simply fix a complete order relation over $\mathcal{L}_\Phi$.

We define a complete order relation over $\mathcal{L}_\Phi$. Let f be a bijective mapping from $\mathcal{L}_\Phi$ to $\{1 \dots |\mathcal{L}_\Phi|\}$. A literal x is smaller than a literal y, noted $x \preceq y$, iff $f(x) \leqslant f(y)$. In this way, each literal of Φ is mapped to a unique natural number. $\preceq$ is a complete order. Now, using this order, we get a unique way to rewrite a 2-CNF formula as a set of B-implications. Let Φ be a 2-CNF formula and $(x \vee y) \in \Phi$, we conjunctively add y (respectively x) to $\beta(x)$ (respectively $\beta(y)$), if $x \prec y$ (respectively $y \prec x$).

Algorithm 2 aims to compute the set of B-implications associated to a 2-CNF formula Φ. It takes a 2-CNF formula Φ and a complete order $\preceq$ over $\mathcal{L}(\Phi)$ as inputs, and provides a set of B-implications $B_{[\vee(\wedge)]}(\Phi)$ as output. In line 1, we initialize $\mathcal{C}(x)$ to the empty set for all the literals of Φ. Following our order relation, in lines $2-9$, we build the set $\beta(x)$ for each literal $x \in \mathcal{L}_\Phi$. Indeed, for each binary clause $(x \vee y) \in \Phi$, y is added to $\mathcal{C}(x)$ (respectively x is added to $\mathcal{C}(y)$) if $x \prec y$ (respectively $y \prec x$).

In lines $10-14$, if according to the chosen ordering, we have $x \prec y$ and $\beta(x)$ contains only one literal, then we try to enhance the compression of the set of B-implications $B_{[\vee(\wedge)]}(\Phi)$. In this case, we add x to the set $\mathcal{C}(y)$, only if it contains at least one literal, and we set $\mathcal{C}(x)$ to the emptyset. In line 16, we return only the B-implications of the form $[x \vee \beta(x)]$ only when $\mathcal{C}(x)$ is not empty.

Assuming that adding an element to a set can be performed in constant time, the worst case complexity of the Algorithm 2 is in $\mathcal{O}(|\Phi| + |\mathcal{L}(\Phi)|)$.

Now, we explain how a transactions database is associated to a 2-CNF formula, using a set of B-implications which we consider as an intermediate representation.

Definition 5 (2-CNF to $\mathcal{D}$). *Let Φ be a 2-CNF formula. The transactions database associated to Φ is defined as $\mathcal{D}^b_\Phi = \{(tid_{x_i}, \beta_i) | x_i \vee \beta_i \in B_{[\vee(\wedge)]}(\Phi)\}$.*

Mining4Binary algorithm.

Let us describe our approach to compact a 2-CNF formula Φ, called Mining4Binary (for reducing the size of a set of

Algorithm 2 B-implications

```
Require: A formula $\Phi$, a complete order $\preceq$ over $\mathcal{L}(\Phi)$
 1: $\mathcal{C}(x) = \emptyset,\ \forall x \in \mathcal{L}(\Phi)$
 2: for $c = (x \vee y) \in \Phi$ do
 3:   if $x \prec y$ then
 4:     $\mathcal{C}(x) \leftarrow \mathcal{C}(x) \cup \{y\}$
 5:   else
 6:     $\mathcal{C}(y) \leftarrow \mathcal{C}(y) \cup \{x\}$
 7:   end if
 8: end for
 9: for $x \in \mathcal{L}(\Phi)$ do
10:   if $\mathcal{C}(x) = \{y\}$ and $|\mathcal{C}(y)| > 1$ then
11:     $\mathcal{C}(y) \leftarrow \mathcal{C}(y) \cup \{x\}$
12:     $\mathcal{C}(x) \leftarrow \emptyset$
13:   end if
14: end for
15: return $\{[x \vee (\bigwedge_{y \in \mathcal{C}(x)} y)] | x \in \mathcal{L}(\Phi) \mid \mathcal{C}(x) \neq \emptyset\}$
```

binary clauses). First, after rewriting Φ as $B_{[\vee(\wedge)]}(\Phi)$, we build the transactions database $\mathcal{D}^b_\Phi$. Then the set of closed frequent itemsets and its associated overlap classes $\mathcal{C}$ are computed. The last step aims to reduce the size of the 2-CNF Φ using a slightly modified version of the Algorithm 1. We only need to add two modifications. First, our 2-CNF compression algorithm takes as input $B_{[\vee(\wedge)]}(\Phi)$ and returns a compressed set of B-implications. Secondly, for an itemset $I = \{y_1, \dots, y_n\}$, in line 4 of the Algorithm 1, we introduce a fresh variable x and we add a B-implication $[\neg x \vee (y_1 \wedge y_2 \wedge \dots \wedge y_n)]$ to $B_{[\vee(\wedge)]}(\Phi)$. This modified algorithm returns a set of compressed B-implications. The last step is a trivial translation of the obtained B-implications to 2-CNF.

Obviously, the compression rate depends on the chosen ordering. Indeed, the intermediate representation (B-implications) is build according to a total order, and the transactions database depends on this intermediary representation.

Example 2. *Let us consider the following 2-CNF Φ:*

$$\begin{array}{rl} \Phi = & (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge \\ & (x_1 \vee x_6) \wedge (x_1 \vee x_7) \wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4) \wedge \\ & (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge (x_2 \vee x_7) \wedge (x_3 \vee x_4) \wedge \\ & (x_3 \vee x_6) \wedge (x_3 \vee x_7) \wedge (x_3 \vee x_5) \wedge (x_4 \vee x_5) \wedge \\ & (x_4 \vee x_6) \wedge (x_4 \vee x_7) \wedge (x_5 \vee x_6) \wedge (x_5 \vee x_7) \wedge \\ & (x_6 \vee x_7) \end{array}$$

Using the complete order relation $x_1 \prec \dots \prec x_7$ over $\mathcal{L}_\Phi$, we can rewrite Φ as the following set of B-implications $B^1_{[\vee(\wedge)]}(\Phi)$:

$$\begin{array}{rl} B^1_{\vee[\wedge]}(\Phi) = & \{[x_1 \vee (x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6 \wedge x_7)], \\ & [x_2 \vee (x_3 \wedge x_4 \wedge x_5 \wedge x_6 \wedge x_7)], \\ & [x_3 \vee (x_4 \wedge x_5 \wedge x_6 \wedge x_7)], \\ & [x_5 \vee (x_6 \wedge x_7)], \\ & [x_6 \vee (x_7)]\} \end{array}$$

The transactions database representation $\mathcal{D}^b_\Phi$ is built from $B^1_{[\vee(\wedge)]}(\Phi)$ by considering only $\beta(x_1), \dots, \beta(x_6)$:

tid	*itemset*					
tid_{x_1}	x_2	x_3	x_4	x_5	x_6	x_7
tid_{x_2}		x_3	x_4	x_5	x_6	x_7
tid_{x_3}			x_4	x_5	x_6	x_7
tid_{x_4}				x_5	x_6	x_7
tid_{x_5}					x_6	x_7
tid_{x_6}						x_7

The itemset mining process is done on the conjunctive part of $B^1_{\vee[\wedge]}(\Phi)$ represented in the transactions database. Setting the minimum support threshold to 4, we get as a frequent itemset $\{x_5, x_6, x_7\}$. Using 2-CNF compression algorithm described above, we can rewrite $B^1_{[\vee(\wedge)]}(\Phi)$ as $B^2_{[\vee(\wedge)]}(\Phi)$:

$$\begin{aligned} B^2_{\vee[\wedge]}(\Phi) = \ & \{[x_1 \vee (x_2 \wedge x_3 \wedge y)] \ , \\ & [x_2 \vee (x_3 \wedge x_4 \wedge y)] \ , \\ & [x_3 \vee (x_4 \wedge y)] \ , \\ & [x_5 \vee (x_6 \wedge x_7)] \ , \\ & [x_6 \vee (x_7)] \ , \\ & [\neg y \vee (x_5 \wedge x_6 \wedge x_7)]\} \end{aligned}$$

Finally a simple encoding of $B^2_{[\vee(\wedge)]}(\Phi)$ as CNF formula leads to following compressed 2-CNF formula:

$$\begin{aligned} \Phi = \ & (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3) & \wedge \\ & (x_2 \vee x_4) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6) \wedge (x_5 \vee x_7) & \wedge \\ & (x_6 \vee x_7) & \wedge \\ & (x_1 \vee y) \wedge (x_2 \vee y) \wedge (x_3 \vee y) \wedge (x_4 \vee y) & \wedge \\ & (x_5 \vee \neg y) \wedge (x_6 \vee \neg y) \wedge (x_7 \vee \neg y) \end{aligned}$$

The substitution of the itemset $\{x_5, x_6, x_7\}$ allows us to reduce the size of the 2-CNF formula Φ. Indeed, the resulting 2-CNF contains 5 binary clauses less.

5.2 Special Case of (Bi-)cliques of Binary Clauses

In [12], J. Rintanen addressed the problem of representing big sets of binary clauses compactly. He particularly shows that constraint graphs arising from practically interesting applications (eg. AI planning) contain big cliques or bi-cliques of binary clauses. An identified bi-clique involving the two sets of literals $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and $\mathcal{Y} = \{y_1, y_2, \ldots, y_m\}$ expresses the propositional formula $\Phi = (x_1 \wedge x_2 \wedge \cdots \wedge x_n) \vee (y_1 \wedge y_2 \wedge \cdots \wedge y_m)$, while a clique involving the literals $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ expresses that at-most one literal from $\mathcal{X}$ is $false$.

Bi-cliques of Binary Clauses.

Let us explain how a bi-clique can be compacted with Mining4Binary method. Let $\Phi = [(x_1 \vee y_1) \wedge (x_1 \vee y_2) \vee \cdots \vee (x_1 \vee y_m)] \ldots [(x_n \vee y_1) \wedge (x_n \vee y_2) \vee \cdots \vee (x_n \vee y_m)]$ a bi-clique of $n \times m$ binary clauses (see Figure 1). Considering the complete order relation defined by: $f(x_i) = i, f(y_j) = n + j$. Using this order relation $B_{[\vee(\wedge)]}(\Phi)$ corresponds exactly to $\{(x_i \vee [y_1 \wedge y_2 \wedge \cdots \wedge y_m]) | 1 \leqslant i \leqslant n\}$. Obviously, the transactions database $\mathcal{D}^b_\Phi$ contains a single closed frequent itemset $\{y_1, y_2, \ldots, y_m\}$. Applying our algorithm leads to the following compact representation of $\Phi' = [\bigwedge_{1 \leqslant i \leqslant n}(x_i \vee z)] \wedge [\bigwedge_{1 \leqslant j \leqslant m}(\neg z \vee y_j)]$. We obtain exactly the same gain as in [12] ($\mathcal{O}(n+m)$ binary clauses and one additional variable).

Cliques of Binary Clauses.

Let $\Phi = \bigwedge_{1 \leqslant i \leqslant n-1}[(x_i \vee x_{i+1}) \wedge \cdots \wedge (x_i \vee x_n)]$ be a clique of n^2 binary clauses (see Figure 2). The set $B_{[\vee(\wedge)]}(\Phi) = \{[x_i \vee (x_{i+1} \wedge \cdots \wedge x_n)] | 1 \leqslant i \leqslant n-1\}$ using the order relation defined by: $f(x_i) = i$. If we take a closer look to D^b_Ψ, the closed frequent itemset I with the greatest value $\alpha(I)$ corresponds to $\{x_{n/2}, \ldots, x_n\}$. In the first $\frac{n}{2}$ rows of D^b_Φ, I is substituted by a fresh variable x and a new set of binary clauses $[x \vee (x_{\frac{n}{2}} \wedge, \cdots \wedge x_n)])$ is added to it, leading to two subproblems of size $\frac{n}{2} + 1$. Obviously, the same treatment is done on the set $B_{[\vee(\wedge)]}(\Phi)$. Consequently, the number of variables is defined by the following recurrence equation:

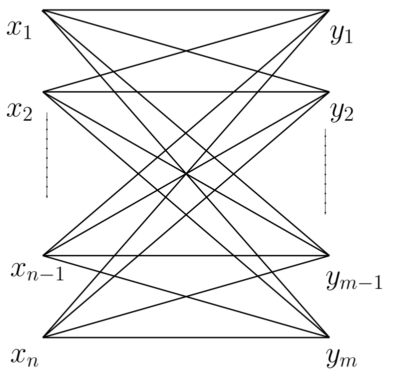

Figure 1: Bi-clique representation of the $n \times m$ clauses

$$\mathcal{V}(n) = 2 \times \mathcal{V}(\frac{n}{2} + 1) + 1 \qquad (1)$$

$$\mathcal{V}(6) = 1. \qquad (2)$$

The basic case is reached for $n = 6$, where the last fresh variable is introduced to represent the conjunction $x_4 \wedge x_5 \wedge x_6$. For $n < 6$ no fresh variable is introduced because no frequent closed itemset can lead to a reduction of the size of the formula. Consequently, from the solution of the previous recurrence equation, we obtain that our encoding is in $\mathcal{O}(n)$ auxiliary variables. Using the same reasoning, we also obtain the same complexity $\mathcal{O}(n)$ for the number of binary clauses. This corresponds to the complexity obtained in [12].

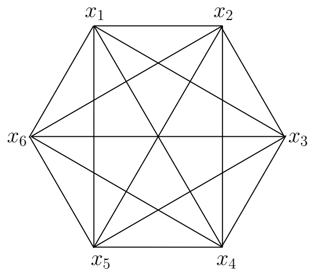

Figure 2: Clique representation of n^2 clauses

The two special cases of cliques and bi-cliques of binary clauses considered in this section, allow us to show that when a constraint is not well encoded, our approach can be used to correct and to derive a more efficient and compact encodings automatically.

6. EXPERIMENTS

In this section, we present an experimental evaluation of our proposed approaches. Two kind of experiments have been conducted. The first one deals with size reduction of arbitrary CNF formulas using Mining4SAT algorithm, while the second one attempts to reduce the size of the 2-CNF sub-formulas only, using Mining4Binary algorithm.

Both algorithms are tested on different benchmarks taken from the last SAT challenge 2012. From the 600 instances

Instance	orig.	comp.	% red
1dlx_c_iq57_a	190 Mb	164 Mb	13.68 %
6pipe_6_ooo.*-as.sat03-413	11 Mb	7.7 Mb	30.00 %
9dlx_vliw_at_b_iq6.*-*04-347	76 Mb	65 Mb	14.47 %
abb313GPIA-9-c.*.sat04-317	21 Mb	6.9 Mb	67.14 %
E05F18	3.7 Mb	2.2 Mb	40.54 %
eq.atree.braun.11.unsat	120 Kb	72 Kb	40.00 %
eq.atree.braun.12.unsat	144 Kb	88 Kb	38.88 %
k2mul.miter.*-as.sat03-355	1.5 Mb	1.3 Mb	13.33 %
korf-15	1.2 Mb	752 Kb	37.33 %
rbcl_xits_08_UNSAT	1.1 Mb	856 Kb	22.18 %
SAT_dat.k45	3.5 Mb	2.6 Mb	25.71 %
traffic_b_unsat	18 Mb	12 Mb	33.33 %
x1mul.miter.*-as.sat03-359	1.1 Mb	928 Kb	15.63 %
9dlx_vliw_at_b_iq3	19 Mb	15 Mb	21.05 %
9dlx_vliw_at_b_iq4	31 Mb	26 Mb	16.12 %
AProVE07-09	2.8 Mb	2.7 Mb	3.57 %
eq.atree.braun.10.unsat	96 Kb	56 Kb	41.66 %
goldb-heqc-frg1mul	348 Kb	328 Kb	5.74 %
minand128	7.7 Mb	2.6 Mb	66.23 %
ndhf_xits_09_UNSAT	2.6 Mb	2.1 Mb	19.23 %
velev-pipe-o-uns-1.1-6	5.5 Mb	4.4 Mb	20.00 %

Table 3: Results of Mining4SAT : a general approach

of the application category submitted to this challenge, we selected 100 instances while taking at least one instance from each family. All tests were made on a Xeon 3.2GHz (2 GB RAM) cluster and the timeout was set to 4 hours.

In Table 3 and Table 4, we indicate the size in Kilobytes (`Kb`) or Megabytes (`Mb`) of each SAT instance before (`orig.`) and after reduction (`comp.`). We also provide *%red*, the reduction percentage.

Table 3 highlights the results obtained by Mining4SAT general approach. In this experiments, and to allow possible reductions, we only search for frequent closed itemsets of size greater or equal to 4. Consequently, binary clauses are not considered. As we can observe, our Mining4SAT reduction approach allows us to reduce the size more than 20% on the majority of instances. Let us also note that the maximum (67.14 %) is reached in the case of the instance *abb313GPIA-9-c.*.sat04-317*: its original size is `21 Mb` and its size after reduction is `6.9 Mb`.

In Table 4, we present a sample of the results obtained by Mining4Binary algorithm on compacting only binary clauses. We observe similar behavior as in the first experiment in terms of size reduction.

To study the influence of our size reduction approaches on the solving time, we also run the SAT solver MiniSAT 2.2 on both the original instance and on those obtained after reduction. As a summary, our approach achieve significant reductions in the size of instances without loosing solving effectiveness.

In our experiments, we have presented the two compression algorithms, Mining4SAT (for clauses of arbitrary size) and Mining4Binary (for 2-CNF formulae). As the two algorithms can be applied independently, we have not presented the results using a two steps compression algorithm: application of the general Mining4SAT algorithm in the first step on the original formula, followed by the specialized Mining4Binary algorithm on the compressed formula derived in the first step. The results can be derived by cumulating the reductions obtained in both steps. For example, if we take the SAT instance `1dlx_c_iq57_a` presented in both Table 3 and Table 4, the reduction by the two steps algorithm is

Instance	orig.	comp.	% red
velev-pipe-o-uns-1.1-6	5.5 Mb	3.2 Mb	41.81 %
9dlx_vliw_at_b_iq2	11 Mb	6 Mb	44.45 %
1dlx_c_iq57_a	190 Mb	124 Mb	34.73 %
7pipe_k	14 Mb	5.4 Mb	61.42 %
SAT_dat.k100.debugged	16 Mb	13 Mb	18.75 %
IBM_FV_2004_rule_batch _2_31_1_SAT_dat.k80.debugged	9.7 Mb	7.5 Mb	22.68 %
sokoban-sequential-p145-*.040-*	24 Mb	14 Mb	41.66 %
openstacks-*-p30_1.085-*	30 Mb	26 Mb	13.33 %
aaai10-planning-ipc5-*-12-step16	17 Mb	12 Mb	29.41 %
k2fix_gr_rcs_w8.shuffled	3.4 Mb	1.7 Mb	50.00 %
homer17.shuffled	20 Kb	16 Kb	20.00 %
gripper13u.shuffled-as.sat03-395	524 Kb	364 Kb	30.35 %
grid-strips-grid-y-3.045-*	52 Mb	42 Mb	19.23 %

Table 4: Results of Mining4Binary: a 2-CNF approach

computed as follows : $(190 - 164) + (190 - 124) = 92Mb$. The reduction ratio is equal to 48.42%.

7. AKNOWLEDGMENTS

We thank the reviewers for their helpful comments. This work has been supported in part by the French ANR project *"DAG: Declarative Approaches for Enumerating Interesting Patterns"* under the Défis program 2009.

8. CONCLUSION AND FUTURE WORKS

In this paper, we propose the first data-mining approach, called Mining4SAT, for reducing the size of Boolean formulae in conjunctive normal form (CNF). It can be seen as a preprocessing step that aims to discover hidden structural knowledge that are used to decrease the number of literals and clauses. Mining4SAT combines both frequent itemset mining techniques for discovering interesting substructures, and Tseitin-based approach for a compact representation of CNF formulae using these substructures. Thus, we show in this work, inter alia, that frequent itemset mining techniques are very suitable for discovering interesting patterns in CNF formulae.

Since we use a greedy algorithm in our approach, the formula obtained after transformation is not guaranteed to be optimal w.r.t. size. An important open question, which we will study in future works, is how to optimally use the closed frequent itemsets ranging in an overlap class. For the special case of sets of binary clauses, finding a better complete ordering leading to an optimal compression is clearly an important challenge.

Finally, our framework can be extended to constraint satisfaction problems (CSP), Pseudo Boolean etc.

9. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 207–216, Baltimore, 1993. ACM Press.

[2] E. Coquery, S. Jabbour, L. Saïs, and Y. Salhi. A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 258–263, 2012.

[3] H. E. Dixon, M. L. Ginsberg, D. K. Hofer, E. M. Luks, and A. J. Parkes. Implementing a generalized version of resolution. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 55–60, 2004.

[4] N. Eén and N. Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[5] M. L. Ginsberg and A. J. Parkes. Search, subsearch and qprop. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, 2000.

[6] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 122–132, 2004.

[7] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, June 2003.

[8] T. Guns, S. Nijssen, and L. D. Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.

[9] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535, 2001.

[10] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[11] L. D. Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *ACM SIGKDD*, pages 204–212, 2008.

[12] J. Rintanen. Compact representation of sets of binary constraints. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 143–147. IOS Press, 2006.

[13] A. Tiwari, R. Gupta, and D. Agrawal. A survey on frequent pattern mining: Current status and challenging issues. *Inform. Technol. J*, 9:1278–1293, 2010.

[14] G. Tseitin. On the complexity of derivations in the propositional calculus. In H. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[15] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In R. J. B. Jr., B. Goethals, and M. J. Zaki, editors, *FIMI*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[16] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 344–353, New York, NY, USA, 2004.

[17] International sat competition. http://www.satcompetition.org. Organized in conjunction with the International Conference on Theory and Applications of Satisfiability Testing.