Ann Math Artif Intell (2017) 80:91–112
DOI 10.1007/s10472-016-9528-4

Local and global symmetry breaking in itemset mining

Belaïd Benhamou[1]

Published online: 15 October 2016

Abstract The concept of symmetry has been extensively studied in the field of constraint programming and in the propositional satisfiability. Several methods for detection and removal of these symmetries have been developed, and their use in known solvers of these domains improved dramatically their effectiveness on a big variety of problems considered difficult to solve. The concept of symmetry may be exported to other areas where some structures can be exploited effectively. Particularly, in the area of data mining where some tasks can be expressed as constraints or logical formulas. We are interested here, by the detection and elimination of *local* and *global* symmetries in the item-set mining problem. Recent works have provided effective encodings as Boolean constraints for these data mining tasks and some idea on symmetry elimination in this area begin to appear, but still few and the techniques presented are often on *global symmetry* that is detected and eliminated statically in a preprocessing phase. In this work we study the notion of *local symmetry* and compare it to *global symmetry* for the itemset mining problem. We show how local symmetries of the boolean encoding can be detected dynamically and give some properties that allow to eliminate theses symmetries in SAT-based itemset mining solvers in order to enhance their efficiency.

Keywords Symmetry · Item-set mining · Data mining · Satisfiability · Constraint programming

✉ Belaïd Benhamou
belaid.benhamou@univ-amu.fr

[1] Aix-Marseille Université, Laboratoire des Sciences de l'information et des Systèmes (LSIS), Domaine Universitaire de Saint Jérôme, Avenue Escadrille Normandie Niemen, 13397 Marseille Cedex 20, France

1 Introduction

The work we propose here is to investigate the notion of *local symmetry*[1] elimination in the Frequent Itemset Mining (FIM) [1] and compare it to *global symmetry*.[2] The itemset mining problem has several applications in real-life problems and remains central in the data mining research field. The most known example is the one considered by large retail organizations called *basket data*. A record of such data contains essentially the customer identification, the transaction date and the items bought by the customer. Advances in bar-codes technology, the use of credit cards of frequent-customer card make it now possible to collect and store a great amount of sale data. It is then important for the retail firms to know the set of items that are frequently bought by customers. This is the frequent itemset problem.

Since its introduction in 1993 [1], several highly scalable algorithms are introduced [2, 18, 23, 26, 36, 44, 45, 47] to enumerate the sets of frequent items. The challenging questions investigated in such algorithms are: in one hand how to compute all the frequent and pertinent item-sets in a reasonable CPU time and in the other hand how to compact the output and reduce its size when there is a huge number of such item sets. Many other data mining tasks exist, such as the association rule mining, the frequent pattern, clustering and episode mining etc, but almost all of them are closely in relationship to the itemset mining which looks to be the principal problem. A lot of efficient and scalable algorithms are developed for target and specific mining tasks. As stated in [42], different methods for the itemset mining are provided. Mainly they differ from each other in the way they explore the search space, the data structure they use, the exploitation of the anti-monotonicity property. The other important point is the size of the output of such algorithms. Some solutions are found, for instance one can enumerate only the closed, the maximal, the condensed, the preferred, or the discriminative item sets instead of all the frequent item sets.

On the other hand, the data mining community introduced the *constraint-based mining* framework in order to specify in terms of constraints the properties of the patterns to be mined [15–17, 38]. A wide variety of constraints are successfully integrated and implemented in different specific data mining algorithms.

Recently De Raedt et al. [24, 40] introduced the alternative of using constraint programming in data mining. They showed that a such alternative can be efficiently applied for a wide range of pattern mining problems. Most of the pattern mining constraint had been expressed in a declarative constraint programming language. This include frequency constraint, closeness, maximality, and anti-monotonic then use a constraint programming system like Gecode as a black box to solve the problem. A strength point here is that different constraints can be combined and solved without the need to modify the solver, unlike in the existing specific data mining algorithms. Since the introduction of this declarative approach, there is a growing interest in finding generic methods to solve data mining tasks. For instance, several works expressed data mining problems as boolean satisfiability problem [27, 30–32, 34, 41] and used efficient modern SAT solvers as black boxes to solve them. More recently, a constraint declarative framework for solving data mining tasks called MininZinc [25], had been introduced.

On the other hand, symmetry is a fundamental property that can be used to study various complex objects, to finely analyze their structures or to reduce the computational complexity

[1] The symmetry of the sub-problems corresponding to the different nodes of the search tree.

[2] The symmetry of the initial problem corresponding to the root of the search tree.

when dealing with combinatorial problems. Krishnamurthy introduced in [33] the principle of symmetry to improve resolution in propositional logic. Symmetries for Boolean constraints are studied in depth in [11, 12]. The authors showed how to detect them and proved that their exploitation is a real improvement for several automated deduction algorithms. Since that, many research works on symmetry appeared. For instance, the static approach used by James Crawford et al. in [19] for propositional logic theories consists in adding constraints expressing the global symmetries of the problem. This technique has been improved in [6] and extended to 0–1 Integer Logic Programming in [7]. The notion of interchangeability in Constraint Satisfaction Problems (CSPs) is introduced in [21] and symmetry for CSPs is studied earlier in [9, 39].

But the notion of symmetry in the field of data mining is not well studied yet. Only few works on *global symmetry* elimination are introduced for some specific data mining algorithms that are targeted to solve some data mining tasks [20, 22, 28, 29, 35, 37, 46].

As far as we know, there is no *local symmetry* breaking method in the framework of data mining. In this work, we investigate dynamic local symmetry detection and elimination and compare it to *global symmetry* exploitation in SAT-based item set mining solvers. *Local symmetry* is the symmetry that we can discover at each node of the search tree during search. *Global symmetry* is the particular local symmetry corresponding to the root of the search tree (the symmetry of the initial problem). Almost all of the known works on symmetry are on global symmetry. Only few works on local symmetry [11, 12] are known in the literature. Local symmetry breaking remains a big challenge.

In most of the data mining tasks, we usually need to enumerate structures. This could turn into a great size output when there is a huge number of such structures and thus could be time consuming. Eliminating symmetry leads to enumerate only the non symmetrical structures, then could provide a more pertinent and compact output and could significantly reduce the CPU time needed to compute the output. In our study case, only non-symmetrical patterns are generated. Patterns are partitioned into symmetrical classes where each symmetrical pattern class is represented by one element. The other elements of the class could be found by applying the considered symmetries on this reference element. Symmetry elimination is also used to avoid exploring in the search tree the symmetrical branches of an explored branch that is shown to gives no solution. These are what we call symmetrical no-goods. We do this because in itemset mining both symmetrical no-goods and symmetrical patterns are in a sens redundant.

The rest of the paper is structured as follows: in Section 2, we give some background on the satisfiability problem, permutations and the necessary notions on itemset mining problem. We study the principle of symmetry in itemset mining in Section 3. In Section 4 we show how symmetries can be detected by means of graph automorphism. We show how local and global symmetries can be eliminated in Section 5. Section 6 shows how symmetry elimination is exploited by a SAT-based item set mining solver. Section 7 shows some related works on symmetry and some experiments on different transaction data-sets are given in Section 8. We conclude the work in Section 9.

2 Background

We summarize in this section some background on the satisfiability problem, permutations, and the necessary notions on the itemset mining problem.

2.1 The propositional satisfiability problem (SAT)

We shall assume that the reader is familiar with the propositional calculus. We give here, a short description. Let V be the set of propositional variables called only variables. Variables will be distinguished from literals, which are variables with an assigned parity 1 or 0 that means $True$ or $False$, respectively. This distinction will be ignored whenever it is convenient, but not confusing. For a propositional variable p, there are two literals: p the positive literal and $\neg p$ the negative one.

A clause is a disjunction of literals $\{p_1, p_2, \ldots, p_n\}$ such that no literal appears more than once, nor a literal and its negation at the same time. This clause is denoted by $p_1 \vee p_2 \vee \ldots \vee p_n$. A set $\mathcal{F}$ of clauses is a conjunction of clauses. In other words, we say that $\mathcal{F}$ is in the conjunctive normal form (CNF).

A truth assignment of a system of clauses $\mathcal{F}$ is a mapping I defined from the set of variables of $\mathcal{F}$ into the set $\{True, False\}$. If $I[p]$ is the value for the positive literal p then $I[\neg p] = 1 - I[p]$. The value of a clause $p_1 \vee p_2 \vee \ldots \vee p_n$ in I is $True$, if the value $True$ is assigned to at least one of its literals in I, $False$ otherwise. By convention, we define the value of the empty clause ($n = 0$) to be $False$. The value $I[\mathcal{F}]$ of the system of clauses is $True$ if the value of each clause of $\mathcal{F}$ is $True$, $False$, otherwise. We say that a system of clauses $\mathcal{F}$ is satisfiable if there exists some truth assignments I that assign the value $True$ to $\mathcal{F}$, it is unsatisfiable otherwise. In the first case, I is called a model of $\mathcal{F}$. Let us remark that a system which contains the empty clause is unsatisfiable.

It is well-known [43] that for every propositional formula $\mathcal{F}$ there exists a formula $\mathcal{F}'$ in conjunctive normal form(CNF) such that $\mathcal{F}'$ is satisfiable iff $\mathcal{F}$ is satisfiable.

2.2 Permutations

Let $\Omega = \{1, 2, \ldots, N\}$ for some integer N, where each integer might represent a propositional variable or an atom. A permutation of Ω is a bijective mapping σ from Ω to Ω that is usually represented as a product of cycles of permutations. We denote by $Perm(\Omega)$ the set of all permutations of Ω and $\circ$ the composition of the permutation of $Perm(\Omega)$. The pair $(Perm(\Omega), \circ)$ forms the permutation group of Ω. That is, $\circ$ is closed and associative, the inverse of a permutation is a permutation and the identity permutation is a neutral element. A pair $(T, \circ)$ forms a sub-group of $(S, \circ)$ iff T is a subset of S and forms a group under the operation $\circ$.

The orbit $\omega^{Perm(\Omega)}$ of an element ω of Ω on which the group $Perm(\Omega)$ acts is $\omega^{Perm(\Omega)}$=$\{\omega^{\sigma} : \omega^{\sigma} = \sigma(\omega), \sigma \in Perm(\Omega)\}$.

A generating set of the group $Perm(\Omega)$ is a subset Gen of $Perm(\Omega)$ such that each element of $Perm(\Omega)$ can be written as a composition of elements of Gen. We write $Perm(\Omega) =< Gen >$. An element of Gen is called a generator. The orbit of $\omega \in \Omega$ can be computed by using only the set of generators Gen.

2.3 The frequent, closed, maximal itemset problem

Let $\mathcal{I} = \{0, \ldots, m-1\}$ be a set of m items and $\mathcal{T} = \{0, \ldots, n-1\}$ a set of n transactions (transaction identifier). A subset $I \subseteq \mathcal{I}$ is called an itemset and a transaction $t \in \mathcal{T}$ over $\mathcal{I}$ is in fact, a pair (t_{id}, I) where t_{id} is the transaction identifier and I the corresponding itemset. In the *basket data* example, the t_{id} of a transaction match the customer identification and I the set of items he put in his basket (he bought). Usually, when there is no confusion, a transaction is just expressed by its identifier. A transaction database $\mathcal{D}$ over

$\mathcal{I}$ is a finite set of transactions such that no different transactions have the same identifier. A transaction database can be seen as a binary matrix $\mathcal{D}(n \times m)$, where $n = \mid \mathcal{T} \mid$ and $m = \mid \mathcal{I} \mid$, with $\mathcal{D}_{t,i} \in \{0, 1\}$ forall $t \in \mathcal{T}$ and $i \in \mathcal{I}$. More precisely, a transaction database is expressed by the set $D = \{(t, I) \mid t \in \mathcal{T}, I \subseteq \mathcal{I}, \forall i \in I : \mathcal{D}_{t,i} = 1\}$. The *coverage* $C_{\mathcal{D}}(I)$ of an itemset I in $\mathcal{D}$ is the set of all transactions in which I occurs. That is, $C_{\mathcal{D}}(I) = \{t \in \mathcal{T} \mid \forall i \in I, \mathcal{D}_{t,i} = 1\}$. The *support* $S_{\mathcal{D}}(I)$ of I in $\mathcal{D}$ is the number $\mid C_{\mathcal{D}}(I) \mid$ of transactions containing I. Moreover, the *frequency* $F_{\mathcal{D}}(I)$ of I in $\mathcal{D}$ is defined by $\frac{|C_{\mathcal{D}}(I)|}{|\mathcal{D}|}$.

Example 1 Consider the transaction database $\mathcal{D}$ made over the set of fruit items $\mathcal{I} = \{Kiwi, Oranges, Apple, Cherries, Plums\}$. For example, we can see in Table 1 that the itemset $I = \{kiwi, Apples\}$ has $C_{\mathcal{D}}(I) = \{1, 3\}$, $S_{\mathcal{D}}(I) = \mid C_{\mathcal{D}}(I) \mid = 2$, and $F_{\mathcal{D}}(I) = 0, 5$.

Given a transaction database $\mathcal{D}$ over $\mathcal{L}$, and θ a minimal support threshold, an itemset I is said to be frequent if $S_{\mathcal{D}}(I) \geq \theta$. I is a closed frequent itemset if in addition to the frequency constraint it satisfies the following constraint: for all itemset J such that $I \subset J$, $S_{\mathcal{D}}(I) > S_{\mathcal{D}}(J)$. I is said to be a maximal frequent itemset if in addition to the frequency constraint it satisfies the following constraint: for all itemset J such that $I \subset J$, $S_{\mathcal{D}}(J) < \theta$. Both closed and maximal itemsets are two known condensed representation for frequent itemsets. The data mining tasks we are dealing with in this work are defined as follows:

Definition 1 1. The frequent itemset mining task consists in computing the following set $\mathcal{FIM}_{\mathcal{D}}(\theta) = \{I \subseteq \mathcal{I} | S_{\mathcal{D}}(I) \geq \theta\}$.
2. The closed frequent itemset mining task consists in computing the following set $\mathcal{CLO}_{\mathcal{D}}(\theta) = \{I \in \mathcal{FIM}_{\mathcal{D}}(\theta) | \forall J \subseteq \mathcal{I}, I \subset J, S_{\mathcal{D}}(I) > S_{\mathcal{D}}(J)\}$.
3. The maximal frequent itemset mining task consists in computing the following set $\mathcal{MAX}_{\mathcal{D}}(\theta) = \{I \in \mathcal{FIM}_{\mathcal{D}}(\theta) | \forall J \subseteq \mathcal{I}, I \subset J, S_{\mathcal{D}}(J) < \theta\}$.

In the next section, we will use the previous definition to express all of the frequent, the closed frequent and the maximal frequent itemset mining tasks as declarative constraints that could be solved by appropriate constraint solvers.

The anti-monotonicity expresses the fact that all the subsets of a frequent itemset are also frequent item sets. More precisely:

Proposition 1 (**Anti-monotonicity**) *If the itemset I is such that $S_{\mathcal{D}}(I) \geq \theta$, then $\forall J \subseteq I$, $S_{\mathcal{D}}(J) \geq \theta$.*

3 Symmetry in itemset mining represented as a satisfiability problem

Both constraint programming and Satisfiability are two known declarative programming frameworks where the user has just only to specify the problem he wants to solve rather than specifying how to solve it. The frequent itemset mining tasks and some of its variants tasks (closed, maximal, ..etc) had been encoded for the first time in [24, 40] as constraint programming tasks where a constraint solver could be used as a black box to solve them. Thereafter, other works [27, 30–32, 34, 41] expressed the data mining tasks as a satisfiability problem where the mining tasks are represented by propositional formulas that are translated into their conjunctive normal forms (CNF) which will be given as inputs to a SAT solver. In

this work we use this last approach to encode the data mining tasks as satisfiability problems in which we detect and eliminate the existing symmetries.

3.1 The boolean encoding of the data mining tasks

Before defining the notion of symmetry, we shall first give the CNF encoding of the data mining tasks. The idea behind the CNF encoding of a data mining task defined on a transaction database $\mathcal{D}$ is to express each of its interpretations as a pair (I, T) where I represents an itemset and T its covering transaction subset in $\mathcal{D}$. To do that, a boolean variable I_i is associated with each item $i \in \mathcal{I}$ and a variable T_t is associated with each transaction $t \in \mathcal{T}$. The itemset I is then defined by all the variables I_i that are true. That is $I_i = 1$, if $i \in I$, and $I_i = 0$ if $i \notin I$. The set of transaction T covered by I is then defined by the set of variable T_t that are true. That is, $T_t = 1$ if $t \in C_{\mathcal{D}}(I)$ and $T_t = 0$ if $t \notin C_{\mathcal{D}}(I)$.

For instance, the $\mathcal{FIM}_{\mathcal{D}}(\theta)$ task can be seen as the search of the set of models $M = \{(I, T) \mid I \subseteq \mathcal{I}, T \subseteq \mathcal{T}, T = C_{\mathcal{D}}(I), \mid T \mid \geq \theta\}$. We have to encode both the covering constraint $T = C_{\mathcal{D}}(I)$ and the frequency constraint $\mid T \mid \geq \theta$. These constraints expressed by the two following boolean and pseudo boolean constraints :

$$\bigwedge_{t \in \mathcal{T}} \left(\neg T_t \leftarrow \bigvee_{i \in \mathcal{I} \mid \mathcal{D}_{t,i}=0} I_i \right)$$

$$\bigwedge_{i \in \mathcal{I}} \left(I_i \rightarrow \sum_{t \in \mathcal{T} \mid \mathcal{D}_{t,i}=1} T_t \geq \theta \right)$$

The formula $\sum_{t \in \mathcal{T}} T_t \geq \theta$ is sufficient to describe the frequency constraint. The formula we propose is an optimization of this constraint. It is called in the community, the reified constraint. This forces the frequency constraint to be verified each time an item is added to the current pattern.

The frequent closed itemset task is specified by adding to the two previous constraints the two following constraint:

$$\bigwedge_{t \in \mathcal{T}} \left(\neg T_t \rightarrow \bigvee_{i \in \mathcal{I} \mid \mathcal{D}_{t,i}=0} I_i \right)$$

$$\bigwedge_{i \in \mathcal{I}} \left(I_i \leftrightarrow \bigwedge_{t \in \mathcal{T} \mid \mathcal{D}_{t,i}=1} T_t \right) \Bigg)$$

The maximal frequent itemset mining is specified by adding to the previous coverage and frequency constraints the following pseudo boolean constraint :

$$\bigwedge_{i \in \mathcal{I}} \left(I_i \leftarrow \sum_{t \in \mathcal{T} \mid \mathcal{D}_{t,i}=1} T_t \geq \theta \right)$$

An important property of these logical encodings established to represent different data mining tasks is that the models of the resulting logical formulas express the solutions of the original data mining tasks considered. This approach is totally declarative, the logical formulas representing the data mining tasks are translated to their equivalent CNF formulas by using known transformation techniques [43] and then given as inputs to a SAT solver which is used as a black box to compute theirs models. For example, if the considered

problem is the search of frequent itemsets in a transaction database $\mathcal{D}$, then the models of the logical formula representing this task in $\mathcal{D}$ express exactly the different frequent itemsets of $\mathcal{D}$ and their covers. That is, if $CNF(k, \mathcal{D})$ denotes the CNF logic encoding of a data mining task k in the transaction database $\mathcal{D}$ and $P^k_{\mathcal{D}}$ a predicate representing the task k in $\mathcal{D}$, then an itemset $I' \subseteq \mathcal{I}$ having $T' \subseteq \mathcal{T}$ as a cover verifies $P^k_{\mathcal{D}}$ $(P^k_{\mathcal{D}}(I', T') = true)$ if I' is an itemset which is an answer to the data mining task k and T' is its cover. Formally, we get the following proposition:

Proposition 2 *Let $J = (I, T)$ be an interpretation of $CNF(k, \mathcal{D})$, $I' = \{i \in \mathcal{I} : I_i = true\}$, and $T' = \{t \in \mathcal{T} : T_t = true\}$, then J is a model of $CNF(k, \mathcal{D})$ iff $P^k_{\mathcal{D}}(I', T') = true$.*

Proof The proof is similar to that one given in [24, 40]. It expresses the fact that the boolean encoding $CNF(k, \mathcal{D})$ is sound. □

3.2 Symmetry in itemset mining

On other hand, the concept of symmetry is well studied in constraint programming and in the satisfiability problem. Since Krishnamurthy's [33] symmetry definition and the one given in [11, 12] in propositional logic, several other definitions are given in the CP community. We will define in the following both the semantic and the syntactic symmetry notions for the boolean encoding of the itemset mining problem and show their relationship.

Definition 2 (**Semantic symmetry**) Let $CNF(k, \mathcal{D})$ be the CNF encoding of the mining task k in $\mathcal{D}$ and $L_{CNF(k,\mathcal{D})}$ its set of literals. A semantic symmetry of $CNF(k, \mathcal{D})$ is a permutation σ defined on $L_{CNF(k,\mathcal{D})}$ such that $CNF(k, \mathcal{D})$ and $\sigma(CNF(k, \mathcal{D}))$ have the same models.

Remark 1 In term of a transaction database $\mathcal{D}$, a semantic symmetry could be seen as an item permutation σ such that $\mathcal{D}$ and $\sigma(\mathcal{D})$ have the same frequent patterns.

In other words a semantic symmetry of $CNF(k, \mathcal{D})$ is a literal permutation that conserves the set of frequent/closed or maximal item sets of $\mathcal{D}$. Semantic symmetry is a general symmetry definition, but its computation is trivially time consuming. We give in the following, the definition of syntactic symmetry that could be computed efficiently and which is considered as a sufficient condition to semantic symmetry.

Definition 3 (**Syntactic symmetry**) Let $CNF(k, \mathcal{D})$ be the boolean encoding of the data mining task k defined on $\mathcal{D}$ and $L_{CNF(k,\mathcal{D})}$ its set of literals. A syntactic symmetry of $CNF(k, \mathcal{D})$ is a permutation σ defined on $L_{CNF(k,\mathcal{D})}$ such that the following conditions hold:

1. $\forall \ell \in L_{CNF(k,\mathcal{D})}, \sigma(\neg \ell) = \neg \sigma(\ell)$,
2. $\sigma(CNF(k, \mathcal{D})) = CNF(k, \mathcal{D})$

In other words, a syntactical symmetry of $CNF(k, \mathcal{D})$ is a literal permutation that leaves $CNF(k, \mathcal{D})$ invariant. If we denote by $Perm(L_{CNF(k,\mathcal{D})}, \circ)$ the group of permutations of $L_{CNF(k,\mathcal{D})}$ and by $Sym(L_{CNF(k,\mathcal{D})}) \subseteq Perm(L_{CNF(k,\mathcal{D})})$ the subset of permutations of

$L_{CNF(k,\mathcal{D})}$ that are the syntactic symmetries of $CNF(k, \mathcal{D})$, then $Sym(L_{CNF(k,\mathcal{D}}, \circ)$ is trivially a sub-group of $Perm(L_{CNF(k,\mathcal{D})}, \circ)$.

Theorem 1 *Each syntactical symmetry of $CNF(k, \mathcal{D})$ is a semantic symmetry of $CNF(k, \mathcal{D})$.*

Proof It is trivial to see that a syntactic symmetry of $CNF(k, \mathcal{D})$ is always a semantic symmetry of $CNF(k, \mathcal{D})$. Indeed, if σ is a syntactic symmetry of $CNF(k, \mathcal{D})$, then $\sigma(CNF(k, \mathcal{D})) = CNF(k, \mathcal{D})$, thus it results that $CNF(k, \mathcal{D})$ and $\sigma(CNF(k, \mathcal{D}))$ have the same models (they express the same item sets satisfying the predicate $P^k_{\mathcal{D}}$). □

Remark 2 The converse of the previous theorems is not true. That is, it is not true that a semantic symmetry of $CNF(k, \mathcal{D})$ is always a syntactical symmetry of of $CNF(k, \mathcal{D})$.

Example 2 Consider the transaction database $\mathcal{D}$ of Table 1 and $k = \mathcal{FIM}_{\mathcal{D}}(\theta)$ for $\theta = 2$. If the set of items $\mathcal{I} = \{Kiwi, Oranges, Apple, Cherries, Plums\}$ are encoded by the scalars $\{1, 2, 3, 4, 5\}$, then the corresponding boolean encoding $CNF(\mathcal{FIM}_{\mathcal{D}}(\theta), \mathcal{D})$ for the frequent item set mining in $\mathcal{D}$ is formed by the set: $var = \{I_1, I_2, I_3, I_4, I_5, T_1, T_2, T_3, T_4\}$ of boolean variables, and the set of clauses:

$$
\begin{aligned}
cl = \{&c_1 : \neg T_1 \vee \neg I_2, c_2 : \neg T_1 \vee \neg I_5,\\
&c3 : \neg T_2 \vee \neg I_1, c4 : \neg T_2 \vee \neg I_5,\\
&c_5 : \neg T_3 \vee \neg I_2, c6 : \neg T_3 \vee \neg I_4,\\
&c7 : \neg T_4 \vee \neg I_1, c8 : \neg T_4 \vee \neg I_4\}
\end{aligned}
$$

and the pseudo boolean constraints:

$$
\begin{aligned}
pb = \{&f_1 : I_1 \rightarrow T_1 + T_3 \geq 2,\\
&f_2 : I_2 \rightarrow T_2 + T_4 \geq 2,\\
&f_3 : I_3 \rightarrow T_1 + T_2 + T_3 + T_4 \geq 2,\\
&f_4 : I_4 \rightarrow T_1 + T_2 \geq 2,\\
&f_5 : I_5 \rightarrow T_3 + T_4 \geq 2\}.
\end{aligned}
$$

The permutation $\sigma = (I_1, I_2)(I_4, I_5)(T_1, T_4)(T_2, T_3)$ defined on the set of variables var is a syntactic symmetry of $CNF(\mathcal{FIM}_{\mathcal{D}}(\theta), \mathcal{D})$.

Table 1 An instance of a transaction database

t_{id}	Itemset
1	Cherries, Apples, Kiwi
2	Cherries, Apples, Oranges
3	Plums, Apples, Kiwi
4	Plums, Apples, Oranges

In the sequel we give some symmetry properties of the boolean encoding $CNF(k, \mathcal{D})$, which express some semantics on the database $\mathcal{D}$.

Definition 4 Two literals I_i and I_j of $CNF(k, \mathcal{D})$ are symmetrical if there exists a symmetry σ of $CNF(k, \mathcal{D})$ such that $\sigma(I_i) = I_j$.

Remark 3 The symmetry between the item literals I_i and I_j expresses the symmetry between the items i and j of $\mathcal{D}$. The previous definition could be applied for the transaction literals T_t to express symmetry between transactions.

Definition 5 The orbit of a literal $I_i \in CNF(k, \mathcal{D})$ on which the group of symmetries $Sym(L_{CNF(k,\mathcal{D})})$ acts is $I_i^{Sym(L_{CNF(k,\mathcal{D})})} = \{\sigma(I_i) : \sigma \in Sym(L_{CNF(k,\mathcal{D})})\}$

Remark 4 All the literals in the orbit of a literal I_i are symmetrical two by two.

Example 3 By considering the symmetry group of Example 2, we can see that the orbit of the item I_1 is $I_1^{Sym(L_{CNF(k,\mathcal{D})})} = \{I_1, I_2\}$ and the one of I_4 is $I_4^{Sym(L_{CNF(k,\mathcal{D})})} = \{I_4, I_5\}$. This means that both *Kiwi* and *Oranges* are two symmetrical items in $\mathcal{D}$. Also, both *Cherries* and *Plums* are two symmetrical items in $\mathcal{D}$. The orbit of I_3 is reduced to itself $I_3^{Sym(L_{CNF(k,\mathcal{D})})} = \{I_3\}$. That is, the item *Apples* is symmetrical with no other item of the transaction database $\mathcal{D}$.

If I is a model of $CNF(k, \mathcal{D})$ and σ a syntactic symmetry, we can get another model of $CNF(k, \mathcal{D})$ by applying σ on the literals which appear in I. These two symmetrical models of $CNF(k, \mathcal{D})$ express two symmetrical patterns of $\mathcal{D}$ and their corresponding covers. Formally we get the following property:

Proposition 3 *I is a model of $CNF(k, \mathcal{D})$ iff $\sigma(I)$ is a model of $CNF(k, \mathcal{D})$.*

Proof Suppose that I is a model of $CNF(k, \mathcal{D})$, then $\sigma(I)$ is a model of $\sigma(CNF(k, \mathcal{D}))$. We can then deduce that $\sigma(I)$ is a model of $CNF(k, \mathcal{D})$ since $CNF(k, \mathcal{D})$ is invariant under σ. The converse can be shown by considering the converse permutation of σ. □

In Example 2, if we consider $\theta = 2$ and the symmetry $\sigma = (I_1, I_2)(I_4, I_5)(T_1, T_4)(T_2, T_3)$, then we can find some symmetrical models in $CNF(k, \mathcal{D})$ (symmetrical frequent item sets in $\mathcal{D}$). For instance, $J = (I, T) = \{I_1, I_3, T_1, T_3\}$ is a model of $CNF(k, \mathcal{D})$ that corresponds to the frequent item set $\{Kiwi, Apples\}$ and its cover {1,3} in $\mathcal{D}$. By the symmetry σ we can deduce that $\sigma(J) = \{I_2, I_3, T_2, T_4\}$ is also a model of $CNF(k, \mathcal{D})$ which corresponds to the frequent itemsets $\{Oranges, Apples\}$ and its cover {2,4}. These are what we call symmetrical models of $CNF(k, \mathcal{D})$ or symmetrical frequent itemsets of $\mathcal{D}$. A symmetry σ transforms each frequent itemset (a model of the CNF encoding) into a frequent itemset and each no-good (not a frequent itemset or not a model of the CNF encoding) into a no-good. In the following we will use this property to eliminate both the symmetrical no-goods and the symmetrical models.

Theorem 2 *Let I_i and I_j be two literals of $CNF(k, \mathcal{D})$ that are in the same orbit with respect to the symmetry group $Sym(L_{CNF(\mathcal{D})})$, then I_i is true in a model of $CNF(k, \mathcal{D})$ iff I_j is true in a model of $CNF(k, \mathcal{D})$.*

Proof If I_i is in the same orbit as I_j then it is symmetrical with I_j in $CNF(k, \mathcal{D})$. Thus, there exists a symmetry σ of $CNF(k, \mathcal{D})$ such that $\sigma(I_i) = I_j$. If I is a model of $CNF(k, \mathcal{D})$ then $\sigma(I)$ is also a model of $\sigma(CNF(k, \mathcal{D})) = CNF(k, \mathcal{D})$, besides if $I_i \in I$ then $I_j \in \sigma(I)$ which is also a model of $CNF(k, \mathcal{D})$. For the converse, consider $I_i = \sigma^{-1}(I_j)$, and make a similar proof. □

Corollary 1 *Let I_i be a literal of $CNF(k, \mathcal{D})$, if I_i is not true in any model of $CNF(k, \mathcal{D})$, then each literal $I_j \in orbit^{\ell} = \ell^{Sym(L_{CNF(k,\mathcal{D})})}$ is not true in any model of $CNF(k, \mathcal{D})$.*

Proof The proof is a direct consequence of Theorem 2. □

Corollary 1 expresses an important property that we will use to break the local symmetries encountered at each node of the search tree of a SAT-based procedure for the itemset mining problem. That is, if a no-good is detected after assigning the value True to the current literal I_i of $CNF(k, \mathcal{D})$, then we compute the orbit of I_i and assign the value false to each literal in it, since by symmetry the value true will not lead to any model of $CNF(k, \mathcal{D})$. Another use of symmetry when a current model is obtained is to avoid to generate its symmetrical models. Such models are not lost, but could be found by applying the considered symmetries on the current model. This could lead to a reduction of the output.

4 Symmetry detection

The most known technique to detect syntactic symmetries for CNF formulas in satisfiability is the one consisting in reducing the considered formula into a graph [3, 7, 19] whose automorphism group is identical to the symmetry group of the original formula. We adapted the same approach here to detect the syntactic symmetries of the boolean encoding $CNF(k, \mathcal{D})$ of a transaction database. That is, we represent the boolean encoding $CNF(k, \mathcal{D})$ of the transaction database $\mathcal{D}$ by a graph $G_{CNF(k,\mathcal{D})}$ that we use to compute the symmetry group of $CNF(k, \mathcal{D})$ by means of its automorphism group. When this graph is built, we use a graph automorphism tool like Saucy [3] to compute its automorphism group which gives the symmetry group of $CNF(k, \mathcal{D})$. Following the technique used in [3, 7, 19], we summarize bellow the construction of the graph which represent the boolean encoding $CNF(k, \mathcal{D})$. Given the encoding $CNF(k, \mathcal{D})$, the associated colored graph $G_{CNF(k,\mathcal{D})}(V, E)$ is defined as follows:

- Each positive item literal I_i of $CNF(k, \mathcal{D})$ is represented by a vertex $I_i \in V$ of the color 1 in $G_{CNF(k,\mathcal{D})}$. The negative literal $\neg I_i$ associated with I_i is represented by a vertex $\neg I_i$ of color 1 in $G_{CNF(k,\mathcal{D})}$. These two literal vertices are connected by an edge of E in the graph $G_{CNF(k,\mathcal{D})}$.
- Each positive transaction literal T_t of $CNF(k, \mathcal{D})$ is represented by a vertex $T_t \in V$ of the color 2 in $G_{CNF(k,\mathcal{D})}$. The negative literal $\neg T_t$ associated with T_t is represented by a vertex $\neg T_t$ of color 2 in $G_{CNF(k,\mathcal{D})}$. These two literal vertices are connected by an edge of E in the graph $G_{CNF(k,\mathcal{D})}$.
- Each positive auxiliary[3] literal ℓ_i of $CNF(k, \mathcal{D})$ is represented by a vertex $\ell_i \in V$ of the color 3 in $G_{CNF(k,\mathcal{D})}$. The negative literal $\neg \ell_i$ associated with ℓ_i is represented by

[3]The literals used to compute the CNF form $CNF(k, \mathcal{D})$.

a vertex $\neg \ell_i$ of color 3 in $G_{CNF(k,\mathcal{D})}$. These two literal vertices are connected by an edge of E in the graph $G_{CNF(k,\mathcal{D})}$.
- Each clause c_i of $CNF(k, \mathcal{D})$ is represented by a vertex $c_i \in V$ (a clause vertex) of color 4 in $G_{CNF(k,\mathcal{D})}$. An edge connects this vertex c_i to each vertex representing each literal of the corresponding clause.
- Each pseudo boolean constraint (cardinality formula) f_i of $CNF(k, \mathcal{D})$ is represented by a vertex $f_i \in V$ (a pseudo boolean constraint vertex) of color 5 in $G_{CNF(k,\mathcal{D})}$. An edge connects this vertex f_i to each vertex representing each literal of the corresponding pseudo boolean constraint.

Remark 5 There is no need to translate the pseudo boolean constraints into CNF formulas to construct the graph and compute the symmetries.

Different colors are assigned to the different categories of literal vertices in the graph $G_{CNF(k,\mathcal{D})}$. This forces the literals in each category to be swapped between them and avoid for example to search for symmetries between items and transactions or between the items and the auxiliary variables that are just used in the boolean encoding to compute its CNF form.

For instance, the graph $G_{CNF(k,\mathcal{D})}$ of the boolean encoding of Example 2 corresponding to the transaction database of Example 1 is given in Fig. 1. We can remark that four colors are used to represent the different vertices of the graph. There is no

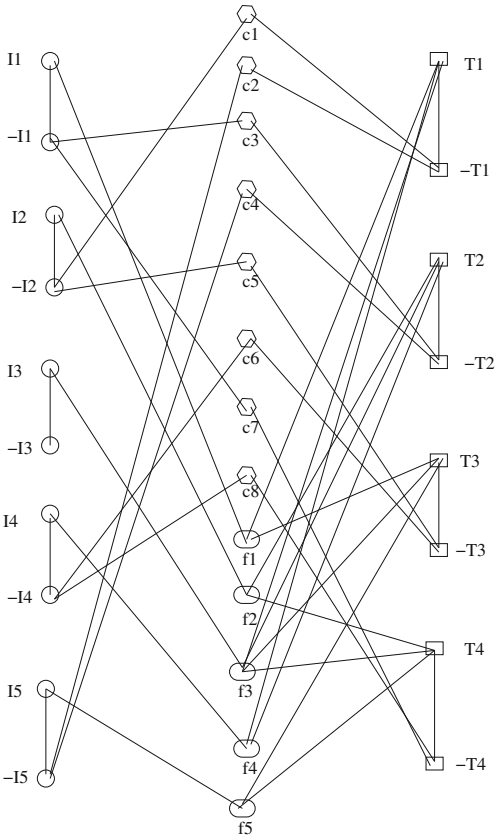

Fig. 1 The graph representation of the boolean encoding of Example 2

need to introduce the fifth color since there is no auxiliary variable in this example. The different colors are represented by different geometrical objects (circles, squares, ellipses,...) in the figure and the negation by the symbol "−". We can remark that $\gamma = (I_1, I_2)(I_4, I_5)(T_1, T_4)(T_2, T_3)(c_1, c_7)(c_2, c_8)(c_3, c_5)(c_4, c_6)(f_1, f_2)(f_4, f_5)$ is an automorphism of the graph $G_{CNF(k,\mathcal{D})}$ from which is deduced the symmetry $\sigma = (I_1, I_2)(I_4, I_5)(T_1, T_4)(T_2, T_3)$ of the boolean encoding $CNF(k, \mathcal{D})$.

An important property of the graph $G_{CNF(k,\mathcal{D})}$ is that it preserves the syntactic group of symmetries of $CNF(k, \mathcal{D})$. That is, the syntactic symmetry group of $CNF(k, \mathcal{D})$ is identical to the automorphism group of its colored graph representation $G_{CNF(k,\mathcal{D})}$. Thus, we could use a graph automorphism system like Saucy on $G_{CNF(k,\mathcal{D})}$ to detect the syntactic symmetry group of $CNF(k, \mathcal{D})$. The graph automorphism system returns a set of generators GEN of the symmetry group from which we can deduce each symmetry of $CNF(k, \mathcal{D})$.

5 Symmetry elimination

5.1 Global symmetry elimination

There are two ways to break symmetries. The first one is to deal with the global symmetries that are present in the formulation of the given problem. Global symmetries can be eliminated in a static way in a pre-processing phase of a SAT-based itemset solver by just adding to the boolean encoding $CNF(k, \mathcal{D})$ the symmetry predicates as it is done in [3, 6, 7, 19].

Given the set of generators of a symmetry group $GEN - Sym(L_{CNF(k,\mathcal{D})}) = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of $CNF(k, \mathcal{D})$ and a total ordering $I_1 < I_2 < \cdots < I_n$ on the variables of $CNF(k, \mathcal{D})$ corresponding to the items of $\mathcal{I}$. The partial lex-leader symmetry-breaking predicate (PLL-SBP) [4] that we have to add to $CNF(k, \mathcal{D})$ is expressed as follows:

$$PP(\sigma_l) = \bigwedge_{1 \le i \le I_i'^{\sigma_l} n} \left[\bigwedge_{1 \le j \le i-1} (I'_j = I_j'^{\sigma_l}) \rightarrow (I'_i \le I_i'^{\sigma_l}) \right]$$

$$PLL - SBP(Sym(L_{CNF(k,\mathcal{D})})) = \bigwedge_{\sigma_l \in GEN - Sym(L_{CNF(k,\mathcal{D})})} PP(\sigma_l)$$

$PP(\sigma_l)$ is the permutation predicate corresponding to the symmetry generator σ_l and the expression $(I'_i \le I_i'^{\sigma_l})$ denotes the clause $(I'_i \rightarrow I_i'^{\sigma_l})$. $I_j'^{\sigma_l}$ is the image value obtained by applying the permutation σ_l on the literal I'_j. To break global symmetry, one needs to add the symmetry breaking predicates $PLL - SBP(Sym(L_{CNF(k,\mathcal{D})})$ to the encoding $CNF(k, \mathcal{D})$ and apply an appropriate SAT solver on the resulting encoding $CNF(k, \mathcal{D}) \wedge PLL - SBP(Sym(L_{CNF(k,\mathcal{D})}))$. Of course a slight modification of the SAT solver is needed to make an enumerator of models (the patterns) and to manage the pseudo boolean constraints.

In summary, for a given data-mining task k, defined on a transaction database $\mathcal{D}$, we first generate the logic encoding $CNF(K, D)$ from which we built the graph $G_{CNF(k,D)}$. Then, we apply the graph automorphism tool Saucy on the generated graph $G_{CNF(k,D)}$ to

calculate the symmetry group of $CNF(K, D)$ that we use to generate the symmetry predicates $PPL - SBP$ that we add to the encoding $CNF(K, D)$. Finally we apply an appropriate SAT solver to the resulting encoding to find the solutions (unsymmetrical patterns).

5.2 Local symmetry elimination

The second way is the elimination of local symmetry that could appear in the sub-problems corresponding to the different nodes of the search tree of a SAT-based itemset solver. Global symmetry can be considered as the particular local symmetry corresponding to the root of the search tree. Local symmetries have to be detected and eliminated dynamically at some decision node of the search tree. Dynamic symmetry detection in satisfiability had been studied in [10–12] where a local syntactic symmetry search method had been given. However, this method is not complete, it detects only one symmetry σ at each node of the search tree when failing in the assignment of the current literal ℓ. As an alternative to this incomplete symmetry search method, a complete method which uses the tool Saucy [3] had been introduced in [10] to detect and break all the syntactic local symmetries of a constraint satisfaction problem (CSP) during search and local symmetry had been detected and eliminated dynamically in a SAT solver [14]. We use the same technique to break local symmetry in itemset mining.

Consider the logic encoding $F = CNF(k, \mathcal{D})$ of the data mining task k defined on the transaction $\mathcal{D}$, and a partial assignment $I = \{\ell_1, \ell_2, \ldots, \ell_i\}$ of a SAT-based itemset solver applied to $CNF(k, \mathcal{D})$. Suppose that $\ell i + 1$ is the current literal under assignment. The assignment I simplifies $CNF(k, \mathcal{D})$ into a sub-formula $F_I = CNF(k, \mathcal{D})_I$ which defines a state in the search space corresponding to the current node n_I of the search tree. The main idea is to maintain dynamically the graph $G_{CNF(k,\mathcal{D})}$ of the sub-formula $F_I = CNF(k, \mathcal{D})_I$ corresponding to the current node n_I, then color the graph $G_{CNF(k,\mathcal{D})_I}$ as shown in the previous section and compute its automorphism group $Aut(CNF(k, \mathcal{D})_I)$. The sub-formula $F_I = CNF(k, \mathcal{D})_I$ can be viewed as the remaining sub-problem corresponding to the unsolved part. By applying an automorphism tool on this colored graph we can get the generator set GEN of the symmetry sub-group existing between literals from which we can compute the orbit of the current literal ℓ_{i+1} that we will use to make the symmetry cut. Figure 2 where we suppose that the orbit of the literal ℓ_{i+1} is $\ell_{i+1}^{Sym(L_{F_I})} = \{\ell'_1, \ell'_2, \ldots, \ell'_m\}$ gives an illustration of the cut.

One possible exploitation of local symmetry is to use Corollary 1 to break dynamically the local symmetries and then prune search spaces of tree search itemset methods. Indeed, if the assignment of the value true to the current literal ℓ_{i+1} defined at the node n_I of the search tree corresponding to the sub-formula F_I is shown to be a failure, then by symmetry, the assignment of the value true to each literal in the orbit of ℓ_{i+1} will result in a failure too. Therefore, the negated literal of each literal in the orbit ℓ_{i+1} is $\ell_{i+1}^{Sym(L_{F_I})} = \{\ell'_1, \ell'_2, \ldots, \ell'_m\}$ has to be assigned the value true in each extension of the partial assignment I. Thus, we prune in the search tree, the sub-space which corresponds to true assignment of the literals of the orbit of ℓ_{i+1}. That is what we call the local symmetry cut. The cut is illustrated in Fig. 2.

The other exploitation of local symmetry is model reduction. We will see in the next subsection that the cut could be applied in the case when a model is obtained in order to eliminate all the symmetrical models of the current model.

5.3 Elimination of symmetrical models (pattern reduction)

Another use of symmetry is the possibility of computing only the non-symmetrical models from which all the other models could be obtained by applying the symmetry group considered.

In the case of global symmetry, both the symmetrical models and the symmetrical no-goods are eliminated by the additional symmetry predicates (SBP) that are added to the boolean encoding $CNF(k, \mathcal{D})$. That is, the interpretations (no-goods or models) are partitioned into symmetrical classes where each class is represented by the least-leader interpretation. All the other interpretations are in a sense redundant then are eliminated.

In the case of local symmetry elimination, the cut we have described above applies even in the dual case when a model instead of a failure is obtained. Indeed, if the interpretation of the current literal ℓ_{i+1} to the value true in the partial interpretation I leads to a model, so we know that we will get a local symmetrical model when interpreting to the value true any symmetrical literal ℓ_j in the orbit of ℓ_{i+1} (theorem 2). In this case, one can avoid generating the local symmetrical model of $I \cup \ell_{i+1}$ by just assigning the value false to ℓ_j when backtracking on the literal ℓ_{i+1}. Thus, by doing this for each literal in the orbit of ℓ_{i+1} we obtain the non-symmetrical models from which we can generate all the others models by applying the calculated symmetries. Local symmetrical models are in a sense redundant, their elimination could help to reduce the size of the output. The symmetry cut is implemented with two options of use: the first one is when a no-good is detected. In this case, the cut is used to eliminate all the local symmetrical no-goods of the current no-good. The other option is when a model is obtained. In this last case the cut is used to eliminate all the local symmetrical models of the considered model.

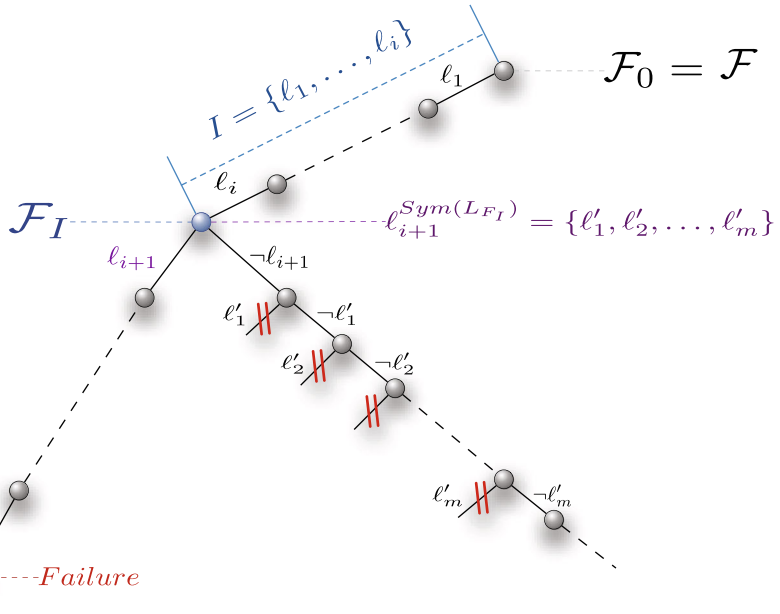

Fig. 2 Local symmetry cut

6 Local symmetry advantage in tree search algorithms

Global symmetry is eliminated statically in a pre-precessing phase by adding the symmetry breaking predicates to the boolean encoding $CNF(k, \mathcal{D})$ and then use a SAT solver on the resulting formula as a black box without any modification.

However, local symmetry must be detected and eliminated dynamically during search. A slight modification of the SAT solver is necessary to integrate local symmetry. We will show how the detected symmetrical literals can be used to increase the efficiency of SAT-based algorithms for the itemset mining. We choose in our implementation the Davis Putnam (DP) procedure to be the baseline method of the solver that we want to improve by the advantage of local symmetry elimination. We will show in the next session how the local symmetry cut had been integrated in a DPLL solver.

If I is a partial interpretation in which the assignment of the value $True$ to the current literal ℓ is shown to be conflicting or a model is obtained, then all the literals in the orbit of ℓ computed by using the group $Sym(L_{CNF(k,\mathcal{D})_I})$ returned by Saucy are symmetrical to ℓ. Thus, we assign the value $False$ to each literal in $\ell^{Sym(L_{CNF(k,\mathcal{D})})}$ since the value $True$ is either contradictory or leads to a symmetrical model, and then we prune the sub-space which corresponds to the value $True$ assignments. The procedure called *Satisfiable* given in Fig. 3 shows how the symmetry cut is integrated in a Davis Putnam procedure.

The input formula $\mathcal{F}$ expresses the boolean encoding $CNF(k, \mathcal{D})$. A *monoliteral* is a unit clause, and a *monotone literal* is a literal that appears either in its positive parity form or in its negative parity form, but not in both them. The function $orbit(\ell, Gen)$ is elementary, it computes the orbit of the literal ℓ from the set of generators Gen returned by Saucy.

In summary, for a given mining task k defined on a transaction database $\mathcal{D}$, we generate as in the case of global symmetries, the logical encoding $CNF(k, D)$ represented by the formula $\mathcal{F}$ in the procedure given in Fig. 3. Next, we apply on the $CNF(k, D)$ encoding a SAT solver including dynamic detection and elimination of local symmetries (Fig. 3) to compute the solutions (non-locally symmetric patterns) and cut the symmetrical no-goods.

```
Procedure SymSatisfiable(F);
begin
    if F = ∅ then F is satisfiable
    else if F contains the empty clause, then F is unsatisfiable
      else begin
        if there exists a mono-literal or a monotone literal ℓ then
          if Satisfiable(F_ℓ) then F is satisfiable
          else F is unsatisfiable
        else begin
          Choose an unsigned literal ℓ of F
          if Satisfiable(F_ℓ) then F is satisfiable
          else
          begin
              Gen=Saucy(F);
              ℓ^{Sym(L_F)}=orbit(ℓ,Gen)={ℓ_1, ℓ_2, ..., ℓ_n};
              if Satisfiable(F_{¬ℓ_1∧¬ℓ_2∧...∧¬ℓ_n}) then F is satisfiable
              else F is unsatisfiable
          end
        end
end
```

Fig. 3 The Davis Putnam procedure with local symmetry elimination

7 Experiments

Now we shall investigate the performances of our search techniques by experimental analysis.

7.1 The input data-sets

We choose for our experiments the following data-sets:

- **Simulated data-sets:** In this class, we use the simulated data-sets, generated specifically to involve interesting symmetries. The process was to create a lot of symmetrical items in different transactions. This will generate totally symmetric Datasets that can be used to see the impact of symmetries in computing only non-symmetrical patterns to reduce the output. The data is available at http://www.cril.fr/decMining.
- **Public datasets:** The datasets used in this class are well known in the data mining community and are available at https://dtai.cs.kuleuven.be/CP4IM/datasets/.

7.2 The experimented methods

Now we shall investigate the performances of our search techniques by experimental analysis. We choose the previous datasets for our study to show the symmetry behavior in solving the itemset mining problem. We expect that symmetry breaking will be profitable in other datasets. Here, we tested and compared three methods:

1. **No-sym:** search without symmetry breaking by using the AVAL solver [8] as the baseline method;
2. **Gl-sym** search with global symmetry breaking. This method uses in pre-processing phase the program SHATTER [5, 7] that detects and eliminates the global symmetries of the considered instance by adding on it symmetry breaking clauses, then apply the solver AVAL [8] to the resulting instance. The CPU time of *Gl-sym* includes the time that SHATTER spends to compute the global symmetry.
3. **Lo-sym:** search with local symmetry breaking. This method implements in AVAL the dynamic local symmetry detection and elimination strategy described in Fig. 3. The CPU time of *Lo-sym* includes local symmetry search time.

The common baseline search method for the three previous methods is AVAL. The complexity indicators are the CPU time (in seconds) and the size of the output. Both the time needed for computing local symmetry and global symmetry are added to the total CPU time of search. The source codes are written in C and compiled on a Core2Duo E8400, 2.8 GHZ and 4 Gb of RAM.

7.3 The obtained results

We reported in Fig. 4 the practical results of the methods: *No-sym*, *Gl-sym*, and *Lo-sym*, on a simulated data *dataset-gen-jss-5* for the closed frequent itemset mining problem. The curves give the CPU times (the ones on the left in the figure) respectively the number of patterns (the ones on the right in the figure) with respect to the minimum support threshold. We can see on the time curves that symmetry elimination is profitable for the itemset mining problem. Indeed, both *Gl-sym* and *Lo-sym* outperform *No-sym*. We also remark that *Lo-sym* detects and eliminates more symmetries than *Gl-sym* and is more efficient. From the curves

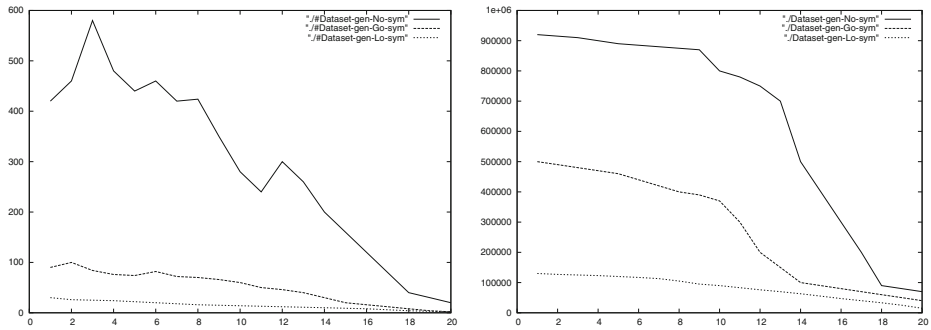

Fig. 4 Results on simulated data (Closed frequent itemsets): CPU time and number of patterns

giving the number of patterns we can see that symmetry leads to significantly decrease the size of the output by keeping only non-symmetrical patterns. We can see that *Lo-sym* reduces more the output than *Gl-sym*. Local symmetry elimination is profitable for solving the itemset mining problem and outperforms dramatically global symmetry breaking on these problems.

In Fig. 5, we reported the practical results of the methods *No-sym* and *Gl-sym* and *Lo-sym* on some public datasets for the frequent itemset mining problem. We can see that there exist some symmetries that are exploited and even the symmetries do not abound, *Gl-sym* and *Lo-sym* outperforms *No-sym* in CPU time. Indeed, many symmetrical no-good branches in the search tree are avoided in the exploration. We can remark that the impact of symmetry decreases as the the frequency threshold increases and local symmetry outperform global symmetry.

In Fig. 6, we reported the results of the methods *No-sym* and *Gl-sym* and *Lo-sym* on both the Australian and Mushroom datasets for the closed frequent itemset mining problem. We can remark in general a diminution of the impact of symmetries in resolution when the frequency threshold increases. The CPU time of each algorithm is reduced in comparison to the frequent pattern case. Indeed, the number of frequent patterns satisfying the additional constraints of closeness decreases. Both global symmetry and local symmetry still profitable in solving this problem variant and local symmetry remains more advantageous than global symmetry.

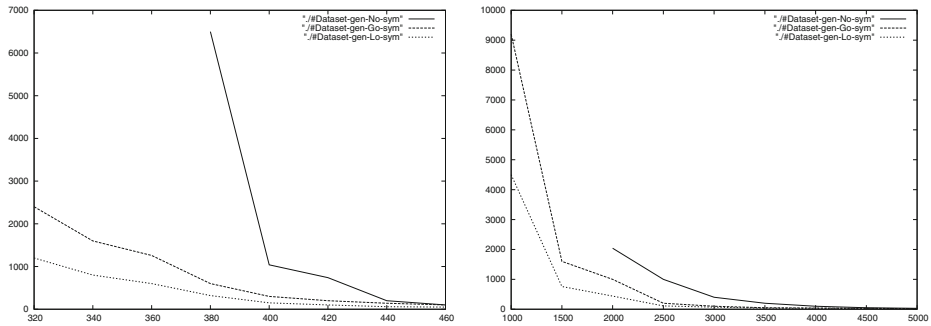

Fig. 5 Results on public data—*Australian and Muchroom*—(frequent itemsets): CPU time

In Fig. 7, we reported the results of the different methods on both the Australian and Mushroom datasets for the Maximal frequent itemset mining problem. We can remark the same behavior as in the case of closed patterns. That is, the number of frequent pattern satisfying the maximality constraint is in a same diminution as in the case of closed patterns. These are two known condensed forms to represent the output. The impact of symmetry decreases as the frequency threshold increases, but both global and local symmetry take advantage in solving this problem variant and local symmetry remains more profitable.

On other hand, no reduction is observed on the number of frequent/closed/Maximal itemsets when using symmetry on the Australian and the Mushroom datasets. On these datasets, most of the computed symmetries involves items in the same transactions. This explains why these particular symmetries does not reduce the number of maximal/closed/frequent itemsets. However, even when the size of the output is not reduced, breaking symmetries using our approach significantly reduce the search space since symmetrical no-goods are not generated.

8 Related works

The purpose of eliminating symmetry in data mining tasks is in general either to obtain a more compact output or to decrease the necessary CPU time for its generation or to handle new mining properties to find interesting frequent patterns. Some symmetry works are introduced in the field of Data mining following this direction.

– Symmetries in graph mining are studied in Desrosiers et al. [20], and in Vanetik [46]. The area of graph mining has a great importance in many applications, but generates a great huge of combinatorial complexity. In Desrosiers et al. [20] symmetry is exploited to prune the search space of sub-graph mining algorithms. However, in Vanetik [46], symmetry is used to find interesting frequent sub-graphs (those having limited diameter and high symmetry). Such graphs represent the more structurally important patterns in all of the the chemical, text and genetic data-sets. Their technique allows also to reduce the necessary CPU to find such graphs.
– Murtagh et al. in [37] used symmetry to get a powerful means of structuring and analyzing massive, high dimensional data stores. They illustrate the powerfulness of hierarchical clustering in case studies in chemistry and finance.

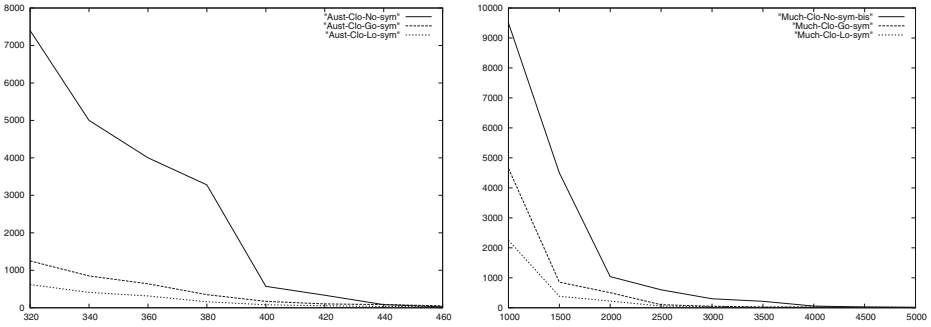

Fig. 6 Results on public data—*Australian and Muchroom*—(Closed itemsets): CPU time

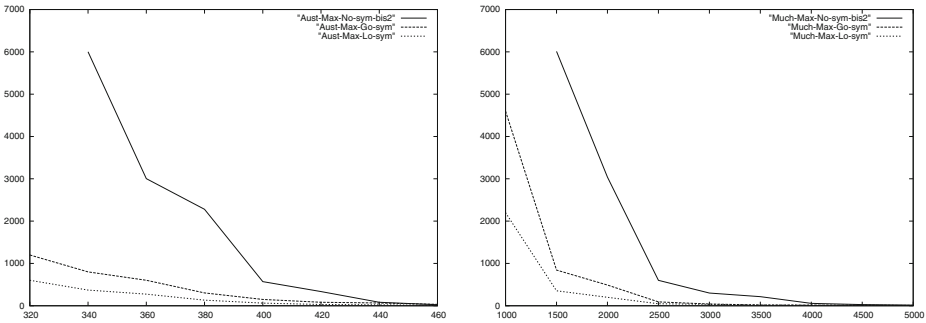

Fig. 7 Results on public data—*Australian and Muchroom*—(Maximal itemsets): CPU time

- Symmetry is also studied in the framework of Zero-BDDs [35]. These symmetries looks very particular, since they are just transpositions of two items and still identity for the remain items. They used such symmetry to studies the properties of symmetrical patterns. Such symmetries are used in [22] to explain in some cases why the number of rules of a minimal cover of a relation is exponential in number of items.
- Two symmetry elimination approaches for frequent itemset mining are introduced in [28]. They consist in rewriting the transaction database in pre-processing phase by eliminating the symmetrical of some items. These symmetries are only some particular global symmetries that the authors use to simplify the underlying transaction in a phase of near-treatment. They do not detect all the global symmetries. These approaches are specific to the data mining task considered. They could be combined with our method as a pretreatment to the database transaction.
- Another approach integrate dynamic symmetry elimination in the Apriori-like algorithm [29] in order to prune the search space of enumerating all the frequent item sets of a transaction database.
- More recently, a method is introduced in [13] to deal with the symmetries between the items of a transaction database. Declarative symmetry breaking predicates are added to the encoding to break such global symmetries. The authors of [13] detect the global symmetry between the items directly from the transaction database. In our approach we detect such symmetry on the boolean encoding of the underlying transaction database in order to compare it to the local symmetry elimination.

All of these approaches deal only with global symmetry. Almost all of them are static techniques that detects symmetry in a pre-processing phase. These are different from the method we developed here which detects and eliminate local symmetry dynamically during the search process.

9 Conclusion

We studied in this work the notions of global and local symmetry for the itemset mining problem expressed as a CNF formulas. We addressed the problem of dynamic symmetry detection and elimination of local symmetry during the search process. That is, the symmetries of each CNF sub-formula defined at a given node of the search tree

and which is derived from the initial formula by considering the partial assignment corresponding to that node. Saucy is adapted to compute this local symmetry by maintaining dynamically the graph of the sub-formula defined at each node of the search tree. Saucy is called with the graph of the local sub-formula as the main input, and then returns the set of generators of the automorphism group of the graph which is shown to be equivalent to the local symmetry group of the considered sub-formula. The proposed local symmetry detection method is implemented and exploited in the DPLL search method to improve its efficiency. Experimental results confirmed that symmetry breaking is profitable for the itemset mining problem expressed as a satisfiability problem.

As a future work, we are looking to eliminate symmetry in other data mining problems like clustering and try to implement some weakened symmetry conditions under which we may detect more symmetries, then experiment it and compare its results with the ones given here.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference n Management of Data, SIGMOD '93, pp. 207–216. ACM, New York (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994)
3. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult SAT instances in the presence of symmetry. In: Proceedings of the 39th Design Automation Conference (DAC 2002), pp. 731–736. ACM Press (2002)
4. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Shatter: efficient symmetry-breaking for boolean satisfiability. In: DAC, pp. 836–839. ACM (2003)
5. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Solving difficult sat instances in the presence of symmetry. In: IEEE Transaction on CAD, vol. 22(9), pp. 1117–1137 (2003)
6. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving difficult instances of boolean satisfiability in the presence of symmetry. IEEE Trans. CAD Integr. Circ. Syst. **22**(9), 1117–1137 (2003)
7. Aloul, F.A., Ramani, A., Markov, I.L., Sakallak, K.A.: Symmetry breaking for pseudo-boolean satisfiabilty. In: ASPDAC'04, pp. 884–887 (2004)
8. Audemard, G., Benhamou, B., Siegel, P.: Aval: an enumerative method for sat. In: Proceedings of the International Conference on Compitational Logic, CL'2000, pp. 373–383. Springer, London (2000)
9. Benhamou, B.: Study of symmetry in constraint satisfaction problems. In: PPCP'94, pp. 246–254 (1994)
10. Benhamou, B., Saïdi, M.R.: Local symmetry breaking during search in csps. In: Springer (ed.) The 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), LNCS, vol. 4741, pp. 195–209. Providence (2007)
11. Benhamou, B., Sais, L.: Theoretical study of symmetries in propositional calculus and application. In: CADE'11, pp. 281–294 (1992)
12. Benhamou, B., Sais, L.: Tractability through symmetries in propositional calculus. J. Autom. Reasoning **12**(1), 89–102 (1994)
13. Benhamou, B., Jabbour, S., Sais, L., Salhi, Y.: Symmetry breaking in itemset mining. In: Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, pp. 86–96 (2014). doi:10.5220/0005078200860096
14. Benhamou, B., Nabhani, T., Ostrowski, R., Saïdi, M.R.: Dynamic symmetry breaking in the satisfiability problem. In: Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR-16. Dakar (2010)
15. Besson, J., Boulicaut, J.F., Guns, T., Nijssen, S.: Generalizing itemset mining in a constraint programming setting. In: Inductive Databases and Constraint-Based Data Mining, pp. 107–126. Springer (2010)

16. Bonchi, F., Lucchese, C.: Extending the state-of-the-art of constraint-based pattern discovery. Data Knowl. Eng. **60**(2), 377–399 (2007)
17. Bucilă, C., Gehrke, J., Kifer, D., White, W.: Dualminer: a dual-pruning algorithm for itemsets with constraints. Data Min. Knowl. Disc. **7**(3), 241–272 (2003)
18. Burdick, D., Calimlim, M., Gehrke, J.: Mafia: a maximal frequent itemset algorithm for transactional databases. In: ICDE, pp. 443–452 (2001)
19. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: Knowledge Representation (KR), pp. 148–159. Morgan Kaufmann (1996)
20. Desrosiers, C., Galinier, P., Hansen, P., Hertz, A.: Improving frequent subgraph mining in the presence of symmetry. In: MLG (2007)
21. Freuder, E.: Eliminating interchangeable values in constraints satisfaction problems. In: AAAI-91, pp. 227–233 (1991)
22. Gély, A., Medina, R., Nourine, L., Renaud, Y.: Uncovering and reducing hidden combinatorics in Guigues-Duquenne bases. In: Ganter, B., Godin, R. (eds.) ICFCA, Lecture Notes in Computer Science, pp. 235–248. Springer (2005)
23. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using fp-trees. IEEE Trans. Knowl. Data Eng. **17**(10), 1347–1362 (2005)
24. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: a constraint programming perspective. Artif. Intell. **175**(12–13), 1951–1983 (2011)
25. Guns, T., Dries, A., Tack, G., Nijssen, S., Raedt, L.D.: Miningzinc: a modeling language for constraint-based mining. In: International Joint Conference on Artificial Intelligence, pp. 1365–1372, Beijing (2013)
26. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD '00, pp. 1–12. ACM, New York (2000)
27. Henriques, R., Lynce, I., Manquinho, V.M.: On when and how to use sat to mine frequent itemsets. CoRR arXiv: 1207.6253 (2012)
28. Jabbour, S., Sais, L., Salhi, Y., Tabia, K.: Symmetries in itemset mining. In: 20th European Conference on Artificial Intelligence (ECAI '12), pp. 432–437. IOS Press (2012)
29. Jabbour, S., Khiari, M., Sais, L., Salhi, Y., Tabia, K.: Symmetry-based pruning in itemset mining. In: 25th International Conference on Tools with Artificial Intelligence (ICTAI'13). IEEE Computer Society, Washington DC (2013)
30. Jabbour, S., Sais, L., Salhi, Y.: Boolean satisfiability for sequence mining. In: CIKM, pp. 649–658 (2013)
31. Jabbour, S., Sais, L., Salhi, Y.: Top-k frequent closed itemset mining using top-k sat problem. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD'13), vol. 146, pp. 131–140. Springer (2013)
32. Khiari, M., Boizumault, P., Crémilleux, B.: Constraint programming for mining n-ary patterns. In: Cohen, D. (ed.) CP, Lecture Notes in Computer Science, vol. 6308, pp. 552–567. Springer (2010)
33. Krishnamurthy, B.: Short proofs for tricky formulas. Acta Inf. **22**(3), 253–275 (1985)
34. Métivier, J.P., Boizumault, P., Crémilleux, B., Khiari, M., Loudni, S.: A constraint language for declarative pattern discovery. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, pp. 119–125. ACM, New York (2012)
35. Minato, S.I.: Symmetric item set mining based on zero-suppressed Bdds. In: Todorovski, L., Lavrac, N., Jantke, K.P. (eds.) Discovery Science, Lecture Notes in Computer Science, vol. 4265, pp. 321–326. Springer (2006)
36. Minato, S.I., Uno, T., Arimura, H.: Fast generation of very large-scale frequent itemsets using a compact graph-based representation (2007)
37. Murtagh, F., Contreras, P.: Hierarchical clustering for finding symmetries and other patterns in massive, high dimensional datasets (2010). CoRR arXiv: 1005.2638
38. Pei, J., Han, J., Lakshmanan, L.V.S.: Pushing convertible constraints in frequent itemset mining. Data Min. Knowl. Disc. **8**(3), 227–252 (2004)
39. Puget, J.F.: On the satisfiability of symmetrical constrained satisfaction problems. In: Kamorowski, J., Ras, Z.W. (eds.) Proceedings of ISMIS'93, LNAI 689, pp. 350–361 (1993)
40. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: KDD, pp. 204–212 (2008)
41. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for data mining and machine learning. In: AAAI (2010)
42. Tiwari, A., Gupta, R., Agrawal, D.: A survey on frequent pattern mining: current status and challenging issues. Inform. Technol. J. **9**, 1278–1293 (2010)
43. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Structures in the Constructive Mathematics and Mathematical Logic, pp. 115–125. H.A.O Shsenko (1968)

44. Uno, T., Asai, T., Uchida, Y., Arimura, H.: Lcm: an efficient algorithm for enumerating frequent closed item sets. In: Proceedings of Workshop on Frequent Itemset Mining Implementations (FIMI 03) (2003)
45. Uno, T., Kiyomi, M., Arimura, H.: Lcm Ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: FIMI (2004)
46. Vanetik, N.: Mining graphs with constraints on symmetry and diameter. In: Shen, H.T., Pei, J., Zsu, M.T., Zou, L., Lu, J., Ling, T.W., Yu, G., Zhuang, Y., Shao, J. (eds.) WAIM Workshops, Lecture Notes in Computer Science, vol. 6185, pp. 1–12. Springer (2010)
47. Zaki, M.J., Hsiao, C.J.: Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Trans. Knowl. Data Eng. **17**(4), 462–478 (2005)