A Mining-Based Compression Approach for Constraint Satisfaction Problems

Said Jabbour and Lakhdar Sais and Yakoub Salhi

CRIL - CNRS, University of Artois, France

Abstract. In this paper, we propose an extension of our Mining for SAT framework to Constraint satisfaction Problem (CSP). We consider n-ary extensional constraints (table constraints). Our approach aims to reduce the size of the CSP by exploiting the structure of the constraints graph and of its associated microstructure. More precisely, we apply itemset mining techniques to search for closed frequent itemsets on these two representation. Using Tseitin extension, we rewrite the whole CSP to another compressed CSP equivalent with respect to satisfiability. Our approach contrast with previous proposed approach by Katsirelos and Walsh, as we do not change the structure of the constraints.

1 Introduction

The table constraint is considered for a long time as particularly important in constraint satisfaction problems (CSP). Indeed, on of the most used formulation of CSP consists in expressing the each constraint in extension or as a relation among variables with associated finite domains. Many research work, consider table constraints as the standard representation. Indeed, any constraint can be expressed using a set of allowed or forbidden tuples. However, the size of these kind of extensional constraints might be exponential in the worst case. In [6], Katsirelos and Walsh proposed for the first time a compression algorithm for large arity extensional constraints. The proposed algorithm attempts to capture the structure that may exist in a table constraint. The authors proposed an alternative representation of the set of tuples of a given relation by a set of compressed tuples. The proposed representation may lead to an exponential reduction in space complexity. However, the compressed tuples may be larger than the arity of the original constraint. Consequently, the obtained CSP do not follow the standard representation of the table constraint. The authors use decision trees to derive a set of compressed tuples.

In this paper, we present a new compression algorithm that combines both itemset mining techniques and Tseitin extension principles to derive a new compact representation of the table constraints. First, we show our previous Mining for SAT approach can be extended to deal with the CSP by considering the constraint graph as a transaction database, where the transactions corresponds to the constraints and items to the variables of the CSP. The closed frequent itemsets corresponds to subset of variables shared most often by the different

constraint of the CSP. Secondly, using extension (auxiliary variables) we show how such constraints can be rewritten while preserving satisfiability. Secondly, we consider each table constraint individually, we derive a new transaction database made of a sequence of tuples i.e. a set of indexed tuples. More precisely, each value of a tuple is indexed with its position in the constraint. By enumerating closed frequent itemsets on such transaction database, we are able to search for the largest rectangle in the table constraint. Similarly, with extension principle, we show how such constraint can be compressed while preserving the traditional representation.

2 Technical background and preliminary definitions

2.1 Frequent Itemset Mining Problem

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 *transaction 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 transaction 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 transaction database in Table 1. Each transaction corresponds to the favorite writers of a library member. For instance, we have $\mathcal{S}(\{Hemingway, Melville\}, \mathcal{D}) = |\{002, 004\}| = 2$ and $\mathcal{F}(\{Hemingway, Melville\}, \mathcal{D}) = \frac{1}{3}$.

tid	itemset
001	$Joyce, Beckett, Proust$
002	$Faulkner, Hemingway, Melville$
003	$Joyce, Proust$
004	$Hemingway, Melville$
005	$Flaubert, Zola$
006	$Hemingway, Golding$

Table 1. An example of transaction database $\mathcal{D}$

Let $\mathcal{D}$ be a transaction 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 [3]. The complexity class $\#P$ corresponds to the set of counting problems associated with a decision problems in NP. For example, counting the number of models satisfying a CNF formula is a $\#P$ problem.

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

Definition 1 (Maximal Frequent Itemset). *Let $\mathcal{D}$ be a transaction 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 transaction 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 $\mathcal{C}(\{Hemingway\}, \mathcal{D}) = \{002, 004, 006\}$ and $\mathcal{C}(\{Hemingway, Melville\}, \mathcal{D}) = \{002, 004\}$. 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 database. Indeed, the frequent itemsets correspond to all the subsets of the closed (resp. maximal) frequent itemsets.

Clearly, 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 [9]. In particular, the problem of counting the number of maximal frequent itemsets is $\#P$-complete (see also [9]).

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 algorithm LCM for mining frequent closed itemsets proposed by Takeaki Uno et al. in [8]. 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).

2.2 Constraint Satisfaction Problems: Preliminary definitions and notations

A constraint network is defined as a tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$. $\mathcal{X}$ is a finite set of n variables $\{x_1, x_2, \ldots, x_n\}$ and $\mathcal{D}$ is a function mapping a variable $x_i \in \mathcal{X}$ to a domain of values $\mathcal{D}(x_i) = \{v_{i_1}, v_{i_2} \ldots v_{i_{d_i}}\}$. We note $d = max\{d_i | 1 \leqslant i \leqslant n\}$ the maximum size of the domains, and $\mathcal{V} = \cup_{x \in \mathcal{X}} \mathcal{D}(x)$ the set of all values. $\mathcal{C}$ is a finite set of m constraints $\{c_1, c_2, \ldots, c_m\}$. Each constraint $c_i \in \mathcal{C}$ of arity k is defined as a couple $\langle scope(c_i), R_{c_i} \rangle$ where $scope(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq \mathcal{X}$ is the set of variables involved in c_i and $R_{c_i} \subseteq \mathcal{D}(x_{i_1}) \times \ldots \times \mathcal{D}(x_{i_k})$ the set of allowed tuples i.e. $t \in R_{c_i}$ iff the tuple t satisfies the constraint c_i. We define the size of the constraint network $\mathcal{P}$ as $|\mathcal{P}| = \sum_{c \in \mathcal{C}} |R_c|$ where $|R_c| = \sum_{t \in R_c} |t|$ and $|t| = |scope(c)|$. A solution to the constraint network $\mathcal{P}$ is an assignment of all the variables satisfying all the constraints in $\mathcal{C}$. A CSP (Constraint Satisfaction Problem) is the problem of deciding if a constraint network $\mathcal{P}$ admits a solution or not.

We denote $t[x]$ the value of the variable x in the tuple t. Let $t_1 = (v_1, \ldots, v_k)$ and $t_2 = (w_1, \ldots, w_l)$ be two tuples (of values or variables), we define the non-commutative operator $\oplus$ by $t_1 \oplus t_2 = (v_1, \ldots, v_k, w_1, \ldots, w_l)$. Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP instance, $c = \langle scope(c), R_c \rangle \in \mathcal{C}$ a constraint and $s = (x_1, \ldots, x_k)$ a sequence of variables such that $Var(s) \subseteq scope(c)$ where $Var(s)$ is the set of variables of s. We denote by $R_c[s]$ the following set of tuples:

$$R_c[s] = \{(t[x_1], \ldots, t[x_k]) \mid t \in R_c\}$$

2.3 Tseitin's Extension principle

To explain the Tseitin principles [7] at the basis of linear transformation of general Boolean formulas to a formula in conjunctive normal form (CNF), let us introduce some necessary definitions and notations. 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.

Tseitin's encoding consists in introducing fresh variables to represent subformulae in order to represent their truth values. Let us 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:

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

$$\wedge (t_3 \rightarrow (z_1 \wedge \cdots \wedge z_n))$$

3 Compressing Table Constraints Networks

In this section, we proposed two compression rules for table constraints networks. The first one is based on the constraint graph aims to reduce the size of the constraint network by rewriting the constraints using the most shared variables. The second compression technique based on the microstructure of the constraint network aims to reduce the size of table constraints by exploiting common subtuples.

3.1 Constraint graph Based Compression

CSP instance as transactions database: We describe the transactions database that we associate to a given constraints network. It is obtained by considering the set of variables as a set of items.

Definition 3. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraints network. The transactions database associated to $\mathcal{P}$, denoted $\mathcal{TD}_{\mathcal{P}}$, is defined over the set of items $\mathcal{X}$ as follows:*

$$\mathcal{TD}_{\mathcal{P}} = \{(tid_c, scope(c)) \mid c \in C\}$$

Constraints Graph Rewriting Rule (CGR): We provide a rewriting rule for reducing the size of a constraints network. It is mainly based on introducing new variables using Tseitin extension principle.

Definition 4 (CGR rule). *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraints network, $s = (x_1, \ldots, x_k)$ a tuple of variables and $\{c_1, c_2, \ldots, c_n\} \subseteq \mathcal{C}$ a subset of n constraints of C such that $\mathcal{V}(s) \subseteq scope(c_i)$ for $1 \leqslant i \leqslant n$. In order to rewrite $\mathcal{P}$, we introduce a new variable $y \notin \mathcal{X}$ and a set $\mathcal{N}$ of l new values such that $\mathcal{V} \cap \mathcal{N} = \emptyset$ and $l = |\bigcap_{i=1}^{n} R_{c_i}[s]|$. Let f be a bijection from $\bigcap_{i=1}^{n} R_{c_i}[s]$ to $\mathcal{N}$. We denote by $\mathcal{P}^g$ the constraint network $\langle \mathcal{X}^g, \mathcal{D}^g, \mathcal{C}^g \rangle$ obtained by rewriting $\mathcal{P}$ with respect to s and f:*

- $\mathcal{X}^g = \mathcal{X} \cup \{y\}$;
- *D^g is a domain function defined as follows: $\mathcal{D}^g(x) = \mathcal{D}(x)$ if $x \in \mathcal{X}$, and $\mathcal{D}^g(y) = \mathcal{N}$.*

– $\mathcal{C}^g = \mathcal{C} \setminus \{c_1, \ldots, c_n\} \cup C'$, *where* $C' = \{c_0, c'_1, \ldots, c'_n\}$ *such that:*
  - $c_0 = \langle (y, x_1, \ldots, x_k), \{(f(a_1, \ldots, a_k), a_1, \ldots, a_k) | (a_1, \ldots, a_k) \in \bigcap_{i=1}^{n} R_{c_i}[s]\}\rangle$
  - $c'_i = \langle (scope(c_i) - s) \oplus (y), \{t[scope(c_i) - s] \oplus (f(t[s])) | t \in R_{c_i}, t[s] \in \bigcap_{j=1}^{n} R_{c_j}[s]\}\rangle$

It is important to note that our rewriting rule, achieve a weak form of pairwise consistency [5]. A constraint network is pairwise consistent (PWC) iff it has non-empty relations and any consistent tuple of a constraint c can be consistently extended to any other constraint that intersects with c.

Definition 5 (Pairwise consistency). *[2,5] Let* $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ *be a constraints network.* $\mathcal{P}$ *is pairwise consistent if and only if* $\forall c_i, \forall c_j \in \mathcal{C}, R_{c_i}[scope(c_i) \cap scope(c_j)] = R_{c_j}[scope(c_i) \cap scope(c_j)]$ *and* $\forall c \in \mathcal{C}$, $R_c \neq \emptyset$.

As pairwise consistency deletes tuples from a constraint relation, some values can be eliminated when they have lost all their supports. Consequently, domains can be filtered if generalized arc consistency (GAC) is applied in a second step.

As a side effect, our CGR rewriting rule maintains some weak form of PWC. Indeed, in Definition 4, when a sub-tuple $t[s] \notin \bigcap_{j=1}^{n} R_{c_j}[s]$, the tuple t is then deleted and do not belong to the new constraint c'_i.

Example 1. Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ be a constraints network, where $\mathcal{X} = \{x_1, \ldots, x_4\}$, $\mathcal{D}(x_1) =, \ldots, = \mathcal{D}(x_4) = \{a, b\}$ and $\mathcal{C} = \{c_1, c_2\}$ where $c_1 = \langle \{x_1, x_2, x_3\}, \{(b, a, a), (a, a, b), (a, b, a)\}\rangle$ and $c_1 = \langle \{x_2, x_3, x_4\}, \{(a, b, a), (b, a, a), (b, a, b)\}\rangle$. Let $s = (x_2, x_3)$ be a tuple of variables such that $s \subset scope(c_1)$ and $s \subset scope(c_2)$.By applying the CGR rule on $\mathcal{P}$, we obtain $\mathcal{P}^g = \langle \mathcal{X}^g, \mathcal{D}^g, \mathcal{C}^g\rangle$ such that:

– $\mathcal{X}^g = \mathcal{X} \cup \{y\}$
– $\forall i (1 \leqslant i \leqslant 4), \mathcal{D}^g(x_i) = \{a, b\}$. We have $\bigcap_{j=1}^{2} R_{c_j}[s] = \{(a, b), (b, a)\}$. We define $f((a, b)) = c, f((b, a)) = d$. Then $\mathcal{D}^g(y) = \{c, d\}$.
– $\mathcal{C}^g = \{c_0, c'_1, c'_2\}$
  - $c_0 = \langle \{y, x_2, x_3\}, \{(c, a, b), (d, b, a)\}\rangle$;
  - $c'_1 = \langle \{x_1, y\}, \{(a, c), (a, d)\}\rangle$ and $c'_2 = \langle \{x_4, y\}, \{(a, c), (a, d), (b, d)\}\rangle$

In this simple example, using one additional variable, we reduce the size of the constraint network from $|\mathcal{P}| = 18$ to $|\mathcal{P}^g| = 16$. As we can observe, the value b can be eliminated by GAC from the domain of x_1.

Necessary and sufficient condition for size reduction Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ be a constraints network, and $s = (x_1, \ldots, x_n) \subseteq \mathcal{X}$ be a sub-tuple of variables corresponding to a frequent itemset I_s of $\mathcal{P}^g$ where the minimal support threshold is greater or equal to k. Let $\{c_1, \ldots, c_k\} \subseteq \mathcal{C}$ be the set of constraints such that $s \subseteq scope(c_i)$ for $1 \leqslant i \leqslant k$. Suppose that the constraints network $\mathcal{P}$ is pairwise consistent, in such a case, all the relations associated to each c_i for $1 \leqslant i \leqslant k$ contain the same number p of tuples. Under such worst case hypothesis, the size of $\mathcal{P}$ can be reduced by at least $r = (n \times p \times k - (p \times k + n \times p + p)$. Let us consider

again the example 1. The reduction is at least $r = (2\times3\times2)-(3\times2+2\times3+3) = 12-15 = -3$. If we consider, the tuple $(b,a,b) \in R_{c_1}$ eliminated by the application of the CGR rule. This results in subtracting 5 from the second term of r. Consequently, we obtain a reduction of 2.

Regarding the value of k, one can see that the compression is interesting when $r > 0$ i.e. $k > \frac{n+1}{n-1}$. Indeed, if $n < 2$ then there is no reduction. 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 constraint network is always reduced when $k \geqslant 4$. We obtain exactly the same condition as in our mining based compression approach of Propositional CNF formula [4]. This is not surprising, as CGR rule is an extension of our Mining4SAT approach [4] to CSP.

Closed vs. Maximal: In [4], we introduced two condensed representations of the frequent itemsets: closed and maximal. 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. The example given in [4], show the benefit that can be obtained by considering frequent closed itemsets. In our Mining for CSP approach we search for frequent closed itemsets.

Compression algorithm: Given a constraint network $\mathcal{P}$, we first search for closed frequent itemsets (set of variables) on $\mathcal{TD}_{\mathcal{P}}$ and then we apply the above rewriting rule on the constraint network using the discovered itemsets of variables. For more details on our algorithm, we refer the reader to the Mining4SAT greedy algorithm [4], where the overlap notion between itemsets are considered. The general compression problem can be stated as follows: given a set of frequent closed itemsets (sub-sequence of variables) and a constraints network, the question is to find an ordered sequence of operations (application of the CGR rule) leading to a CSP of minimal size.

3.2 Microstructure Based Compression

In this section, we describe our compression based approach of Table constraints. First, we show how a Table constraint c can be translated to a transaction database $\mathcal{TD}_c$. Secondly, we show how to compress c using itemset mining techniques.

Table constraint as transactions database: Obviously, a table constraint c can be translated in a naive way to a transaction database $\mathcal{TD}_c$. Indeed, one can define the set of items as the union of the domains of the variables in the scope of c ($\mathcal{I} = \cup_{x\in scope(c)}\mathcal{D}(x)$) and a transaction (tid,t) as the set of values involved in the tuple $t \in R_c$. This naive representation is difficult to exploit in our context. Let $I = \{a,b,c\}$ be a frequent itemset of $\mathcal{TD}_c$. As the variables in

each transaction (or tuple) associated to the values in I are different, it is difficult to compress the the constraint while using both classical tuples and compressed tuples [6]. To overcome this difficulty, we consider tuples as sequence, where each value is indexed by its position in the tuple.

Definition 6 (Indexed tuples). *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraint network, and $c_i \in \mathcal{C}$ a table constraint such that $scope(c_i) = (x_{i_1}, x_{i_2}, \ldots, x_{i_{n_i}})$. Let $t \in R_{c_i}$ a tuple of c_i. We define $indexed(t) = (t[x_{i_1}]^1, t[x_{i_2}]^2, \ldots, t[x_{i_{n_i}}]^{n_i})$ as an indexed tuple associated to t i.e. each value of the tuple is indexed with its position in the tuple.*

Definition 7 (Inclusion, index). *Let c be a table constraint with $scope(c) = \{x_1, \ldots, x_n\}$ and $t = (v_1, \ldots, v_n) \in R_c$ a tuple of c. We say that $s = (w_1, \ldots, w_k)$ is a sub-tuple of t, denoted $s \subseteq t$, if $\exists 1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$ such that $w_1 = v_{i_1}, \ldots, w_k = v_{i_k}$. We define $index(t) = \{1, \ldots, n\}$, while $index(w) = \{i_1, \ldots, i_k\}$. We also define $vars(index(t)) = scope(c)$ and $vars(index(w)) = \{x_{i_1}, \ldots, x_{i_k}\}$.*

Definition 8. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraints network, and $c \in \mathcal{C}$ a table constraint where $scope(c) = \{x_1, \ldots, x_n\}$. The transaction database associated to c, denoted $\mathcal{TD}_c$, is defined over the set of items $\mathcal{I} = \bigcup_{t \in R_c} \{t[x_1]^1, \ldots, t[x_n]^n\}$ as follows:*

$$\mathcal{TD}_c = \{(tid_t, indexed(t)) | t \in R_c\}$$

Example 2. Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraints network, where $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{D}(x) = \mathcal{D}(y) = \mathcal{D}(z) = \mathcal{D}(t) = \{a, b\}$. Let $c \in \mathcal{C}$ a table constraint, such that $scope(c) = \{x_1, x_2, x_3, x_4\}$ and $R_c = \{(a,b,b,a), (a,a,b,b), (a,b,a,a), (b,b,a,a), (b,b,b,a)\}$. The transaction database $\mathcal{TD}_c$ associated to c is defined as follows:

tid	itemset
001	a^1, b^2, b^3, a^4
002	a^1, a^2, b^3, b^4
003	a^1, b^2, a^3, a^4
004	b^1, b^2, a^3, a^4
005	b^1, b^2, b^3, a^4

Table 2. $\mathcal{TD}_c$ a transaction database associated to c

Let $I = \{b^2, a^4\}$ be an itemset of $\mathcal{TD}_c$. We have $\mathcal{S}(I, \mathcal{TD}_c) = |\{001, 003, 004, 005\}| = 2$, $index(I) = \{2, 4\}$ and $vars(index(I)) = \{x_2, x_4\}$.

Microstructure Rewriting Rule (MRR): We now provide a rewriting rule for reducing the size of a table constraint.

Definition 9 (MRR rule). *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a constraints network and $c \in \mathcal{C}$ be a table constraint with $scope(c) = \{x_1, x_2, \dots x_n\}$ and $|R_c| = m$. Let $I = \{v_1^{i_1}, \dots, v_k^{i_k}\}$ be an itemset of $\mathcal{TD}_c$ and $Y = vars(index(I)) = \{x_{i_1}, \dots, x_{i_k}\}$. In order to rewrite c using I, we introduce a new variable $z \notin \mathcal{X}$ and a set $\mathcal{N}$ of l new values such that $\mathcal{V} \cap \mathcal{N} = \emptyset$ and $l = |\bigcup_{t \in R_c} t[Y]|$. Let f be a bijection from $\bigcup_{t \in R_c} t[Y]$ to $\mathcal{N}$. We denote by $\mathcal{P}^m$ the constraints network $\langle \mathcal{X}^m, \mathcal{D}^m, \mathcal{C}^m \rangle$ obtained by rewriting c with respect to I and f:*

- $\mathcal{X}^m = \mathcal{X} \cup \{z\}$;
- D^m *is a domain function defined as follows:* $\mathcal{D}^m(x) = \mathcal{D}(x)$ *if* $x \in \mathcal{X}$, *and* $\mathcal{D}^m(z) = \mathcal{N}$.
- $\mathcal{C}^m = C \setminus \{c\} \cup C'$, *where* $C' = \{c_0, c'\}$ *such that:*
  - $c_0 = \langle (z, Y), \{(f(a_1, \dots, a_k), a_1, \dots, a_k) | (a_1, \dots, a_k) \in \bigcup_{t \in R_c} t[Y]\} \rangle$
  - $c' = \langle (scope(c) - Y) \oplus (z), \{t[scope(c) - Y] \oplus (f(t[Y])) | t \in R_c\} \rangle$

Example 3. Let us consider again the example 2. Applying the MR rewriting rule to c with respect to $I = \{b^2, a^4\}$, and f where $f((b, a)) = c_1$ and $f((a, b)) = c_2$, we obtain the following two constraints:

- $c_0 = \langle \{z, x_2, x_4\}, \{(c_1, b, a), (c_2, a, b)\} \rangle$;
- $c' = \langle \{x_1, x_3, z\}, \{(a, b, c_1), (a, b, c_2), (a, a, c_1), (b, a, c_1), (b, b, c_1)\} \rangle$

It is easy to see that in example 3, applying MRR rule leads to a constraint of greater size. In what follows, we introduce a necessary and sufficient condition for reducing the size of the table constraint.

Necessary and sufficient condition for size reduction Let c be a table constraint, p the number of tuples in R_c, and $s = (v_1, \dots, v_n) \subseteq \mathcal{X}$ be a sub-tuple of values corresponding to a frequent itemset I_s of $\mathcal{TD}^c$ where the minimal support threshold is greater or equal to k. Let $\{t_1, \dots, t_k\}$ be the set of tuples such that $t_i[vars(index(s))] = s$ for $1 \leqslant i \leqslant k$. The size of R_c can be reduced by at least $r = (n \times k - (p + 1 + n + (p - k))$. Let us consider again the example 3. The reduction is at least $r = (2 \times 4 - (5 + 1 + 2 + (5 - 4)) = 8 - 9 = -1$. In this example, we increase the size of c by one value. Indeed, $|R_c| = 20$ and $|R_{c_0}| + |R_{c'}| = 6 + 15 = 21$.

Regarding the value of k, one can see that applying MRR rule is interesting when $r > 0$ i.e. $k > \frac{2 \times p + n + 1}{n + 1}$. In the previous example, no reduction is obtained as $4 > \frac{2 \times 5 + 2 + 1}{2 + 1}$. ($4 > 4$, the condition is not satisfied).

Compression algorithm of a table constraint: Given a constraint network $\mathcal{P}$, and c a constraint table of $\mathcal{P}$, we first search for closed frequent itemsets (sub-tuple of values) on $\mathcal{TD}_c$ and then we apply the above rewriting rule on the table constraint using the discovered itemsets of values. Similarly to the constraint graph based compression algorithm, our microstructure based compression algorithm can be derived from the one defined in [4].

As a summary, to compress general CSP, our approach first apply constraint graph based compression algorithm followed by the microstructure based compression algorithm.

4 Conclusion and Future Works

In this paper, we propose a data-mining approach, called Mining4CSP, for reducing the size of constraints satisfaction problems when constraints are represented in extension. It can be seen as a preprocessing step that aims to discover hidden structural knowledge that are used to decrease the size of table constraints. Mining4CSP combines both frequent itemset mining techniques for discovering interesting substructures, and Tseitin-based approach for a compact representation of Table constraints using these substructures. Our approach is able to compact a CSP by considering both its associated constraint graph and microstructure. This allows us to define a two step algorithm. The first step, named coarse-grained compression, allows to compact the constraint graph using patterns representing subsets of variables. The second step, named fine-grained compression allows us to compact a given set of tuples of a given table constraint using patterns representing subset of values. Finally, an experimental evaluation on CSP instances is short term perspective.

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. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.
3. Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, June 2003.
4. Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. Mining to compact cnf propositional formulae. *CoRR*, abs/1304.4415, 2013.
5. P. Janssen, P. Jégou, B. Nouguier, and M.C. Vilarem. A filtering process for general constraint satisfaction problems: Achieving pairwise consis- tency using an associated binary representation. In *Proceedings of IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.
6. George Katsirelos and Toby Walsh. A compression algorithm for large arity extensional constraints. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 379–393. Springer, 2007.
7. G.S. Tseitin. On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.
8. Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors, *FIMI*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
9. Guizhen 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. ACM.