Exploiting Data Mining Techniques for Compressing Table Constraints
Soufia Bennai, Kamal Amroun, Samir Loudni

To cite this version:
Soufia Bennai, Kamal Amroun, Samir Loudni. Exploiting Data Mining Techniques for Compressing Table Constraints. IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Nov 2019, Portland, Oregon, United States. 10.1109/ICTAI.2019.00015 . hal-02463468

HAL Id: hal-02463468
https://hal-normandie-univ.archives-ouvertes.fr/hal-02463468
Submitted on 5 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Exploiting Data Mining Techniques for Compressing Table Constraints

Soufia Bennai  
LIMED – Faculty of Exact Sciences, University of Bejaia  
Bejaia, Algeria  
sofia.bennai21@gmail.com

Kamal Amroun  
LIMED – Faculty of Exact Sciences, University of Bejaia  
Bejaia, Algeria  
kamalamroun@gmail.com

Samir Loudni  
GREYC (CNRS UMR 6072), University of Caen Normandy  
Caen, France  
samir.loudni@unicaen.fr

Abstract—In this paper, we propose an improvement of the compression step of sliced table method proposed by Gharbi et al. [1] for compressing and solving table constraints. We consider only n-ary CSP defined in extensional form. More precisely, we propose to use the cover of an itemset in the FP-tree instead of its frequency to improve the construction step of the resulting compressed tables. Moreover, we propose to exploit the compression rate metric instead of savings to compute frequent itemsets relevant for compression. This allows higher compression and leads to an efficient resolution of compressed tables by identifying more accurate frequent itemsets necessary for compression. Experimental results show the effectiveness and efficiency of our approach.

I. INTRODUCTION

Many real world applications can be formulated as Constraint Satisfaction Problems (CSPs). A CSP can be defined as a set of variables with finite domains of possible values and a set of constraints defined on subsets of variables. Constraints defined in extension (table constraints) are widely used (database, configuration problems, etc.). A list of tuples of allowed or forbidden values of a set of variables is given. The size of table constraints can be very large, especially with the advent of Big Data. So, very large table constraints can be an obstacle for the solving process. Many works have been proposed in the literature to compress CSP in order to facilitate their solving. We can cite Sliced tables [1], Multi-valued Decision Diagrams (MDD) [5], micro-structure based compression [4], etc.

In this work, we present an improvement of the compression step of sliced table method (called in this work Frequent Pattern Tree Compression Method (FPTCM)) which uses an FP-Tree structure to enumerate the frequent itemsets relevant for compressing table constraints [1]. Mainly, we proceed as follows:

- First, we construct the FP-Tree in another way. Instead of considering only the frequencies of items in the header table and in the FP-Tree, we also exploit the notion of coverage which consists on all the tuples in which an item appears. This allows to reduce the time of construction of the compressed relation.
- Second, we propose to compute the compression rate possible with an itemset $u$ instead of its savings $(|u|*(f−1)$ where $f$ is the frequency of $u$) to decide whether $u$ is relevant for compression or not. Because computing the savings of a node and its parent does not consider the same number of tuples.

We evaluated our proposition on some benchmarks, downloaded from https://www.cril.univ-artois.fr/ lecoutre/benchmarks, and the obtained results are very promising.

The rest of this paper is organized as follows. In Section II we give some definitions in Constraint Satisfaction Problems (CSPs) and frequent itemsets mining. Section III reviews some related works. In Section IV, we present the compression step of the sliced table method proposed by Gharbi et al. [1]. Mainly, we also give its weakness. Section V is devoted to our proposition. Experiments carried out in this work are presented in Section VI. We conclude with some remarks and avenue for future works.

II. BACKGROUND

A. Constraint Satisfaction problem

A CSP [2] consists of a finite set of variables $V=\{v_1, \ldots, v_n\}$ with finite domains $D=\{D_1, \ldots, D_n\}$ such that each $D_i$ is the set of values that can be assigned to $v_i$, and a finite set of constraints $C$. Each constraint $c_i \in C$ is defined as a pair $(S(c_i), R(c_i))$, where $S(c_i)$ (the scope of the constraint $c_i$) is the set of variables involved in $c_i$ and $R(c_i)$ is a relation that defines the set of tuples allowed for the variables of $c_i$. We define the arity of $c_i$ as the size of its scope. The objective is to find an assignment $(v_i = d_i)$ with $d_i \in D_i$ for $i = 1, \ldots, n$, such that all constraints are satisfied.

Example 1: Let be the following CSP:

$V = \{v_0, \ldots, v_5\}$, $D = \{D_0, \ldots, D_5\}$ where,

$D_0 = D_1 = D_2 = D_3 = D_4 = D_5 = \{1, \ldots, 7\}$,

$C = \{c_0\}$ where, $c_0 = \{(v_0, v_1, v_2, v_3, v_4, v_5), R(c_0)\}$ and $R(c_0) = \{(1\ 2\ 1\ 1\ 1), (1\ 2\ 1\ 2\ 1), (1\ 2\ 1\ 3\ 1), (1\ 3\ 1\ 1\ 1), (1\ 3\ 2\ 1\ 1), (1\ 3\ 1\ 2\ 1), (1\ 3\ 1\ 3\ 1), (1\ 3\ 1\ 3\ 1)\}$. 


Frequent itemset mining

Let \( \mathcal{I} \) be a set of \( n \) distinct literals called items, an itemset (or pattern) is a non-null subset of \( \mathcal{I} \). The language of itemsets corresponds to \( \mathcal{L}_\mathcal{I} = 2^\mathcal{I} \setminus \emptyset \). A transactional dataset is a multi-set of \( m \) itemsets of \( \mathcal{L}_\mathcal{I} \). Each itemset, usually called a transaction or object, is a dataset entry.

Let \( \mathcal{T} \) a transaction dataset, \( p \in \mathcal{L}_\mathcal{I} \) be an itemset, and \( \text{match}^1 : \mathcal{L}_\mathcal{I} \times \mathcal{L}_\mathcal{I} \mapsto \{\text{true}, \text{false}\} \) a matching operator. The cover of \( p \) w.r.t. \( \mathcal{T} \), denoted by \( \text{cov}(p) \), is the set of transactions in \( \mathcal{T} \) that \( p \) matches: \( \text{cov}(p) = \{t \in \mathcal{T} \mid \text{match}(p, t) = \text{true}\} \).

The frequency of \( p \) is the size of its cover: \( \text{f}(p) = |\text{cov}(p)| \).

Let \( S_{\text{min}} \) be a minimal support threshold. The frequent itemset mining problem consists in computing the set of all patterns \( p \) having a number of occurrences in the dataset exceeding \( S_{\text{min}} : \text{f}(p) \geq S_{\text{min}} \).

Constraint based compression by itemset mining

In this subsection we show how a constraint relation \( R(c_i) \) defined over a constraint \( c_i \) can be represented as a transactional dataset \( \mathcal{T}D_{c_i} \). Then, we show how to compress \( R(c_i) \) using itemset mining techniques.

Let \( \mathcal{P} = (V, \mathcal{D}, \mathcal{C}) \) be a CSP and \( R(c_i) \) be a constraint relation associated with a constraint \( c_i \in \mathcal{C} \). The transactional dataset \( \mathcal{T}D_{c_i} \) is defined as follows: (i) the union of the domains of the variables in the scope of \( c_i \) represent the set of items of \( \mathcal{I} \), (ii) the set of values involved in the tuple \( t \in R(c_i) \) forms a transaction in \( \mathcal{T} \). Table I shows the transactional dataset associated with the constraint relation of \( R(c_0) \) of Example 1.

In this context, an itemset \( p \) of a constraint relation \( R(c_i) \) is an assignment of some variables involved in the scope of \( c_i \).

Example 2: In Table Ia, \( u = \{v_0 = 1, v_2 = 1\} \) is an itemset. The cover of \( u \) is \( \text{cov}(u) = \{t_0, t_1, t_2, t_3, t_5, t_6, t_7\} \) and \( \text{f}(u) = 7 \).

The main idea behind the use of pattern mining to derive a compact representation of the constraint relation is to use frequent itemsets extracted from the transaction dataset as a summary of a set of transactions. These transactions are replaced by each frequent itemset that covers them. The resulting compressed constraint relation consists of a set of entries where each entry contains an itemset and its corresponding sub-table.

Definition 1: The sub-table \( S_t \) associated with an itemset \( u \) of a constraint \( c_i \) is the set of tuples of \( R(c_i) \) containing \( u \) after removing \( u \) from them.

Definition 2: An entry for a constraint relation \( R(c_i) \) is a pair \( (u, S_t) \) such that \( u \) is a frequent itemset and \( S_t \) its corresponding sub-table.

Example 3: Table Ib shows the entry corresponding to the itemset \( u = \{v_0 = 1, v_2 = 1\} \) and its resulting sub-table.

Let \( u \) be a frequent itemset, \( f \) its frequency and \( T \) the set of compressed transactions. Let \( \text{size}_a \) (resp. \( \text{size}_b \)) be the size of \( T \) after (resp. before) compression. To assess the quality of a summary \( S_t \) of a set of transactions \( T \), we define the following metric:

Definition 3 (Compression rate): The compression rate of a set of transactions \( T \) w.r.t. itemset \( u \) is defined as follows:

\[
\text{Rate} = 1 - \frac{\text{size}_a}{\text{size}_b}
\]

where \( \text{size}_b = \text{arity} \times f \).

III. RELATED WORKS

Katsirelos and Walsh [3] proposed a first approach for compressing large arity table constraints using decision trees. The tuples of the original table constraints are replaced by a set of compressed tuples, leading to a more compact representation. Jabbour et al. [4] proposed a SAT based approach for compressing table constraints of a CSP. They proposed two new rewriting rules for reducing the size of the constraint network as well as the size of the constraint relations while preserving the original structure of table constraints. They used closed itemsets to compute a summary of tuples of each table constraint. Another form of compression that uses Multi-valued Decision Diagrams (MDD) was proposed by Cheng et al. [5]. It enumerates the frequent itemsets and replaces each occurrence of an itemset by a unique symbol. The frequent itemsets are independent of their initial position in the tuples. Mairy et al. [9] proposed a new compressed form of table constraints called smart tables which is a set of smart tuples. Smart tuples contain simple arithmetic constraints. Nightingale et al. [10] also proposed a new representation of table constraints called smart tables.
constraints called short tables, where the authors identify short supports that are a generalization of the supports of tuples.

Finally, the authors of [1] introduced a new compression method based on FP-Tree structure to enumerate the frequent itemsets necessary for compressing table constraints. They proposed to use a saving function to decide if a frequent itemset is needed for compression or not. The compressed table constraint consists of a set of entries where each entry contains an itemset and its corresponding sub-table. Our approach improves the method proposed by [1] in several ways (see Section V).

IV. FREQUENT PATTERN TREE COMPRESSION METHOD

In this section we briefly describe the approach of Gharbi et al. [1], called FPTCM (Frequent Pattern Tree Compression Method), for compressing table constraints. Then, we discuss its main weaknesses.

A. FPTCM in nutshell

The principle of FPTCM is to enumerate only frequent itemsets relevant for compression. Such itemsets are obtained as follows: first, FPTCM computes the frequency \( f \) of each item, then it sorts each tuple in decreasing order of frequency values. Items which have a frequency below the threshold \( S_{min} \) are removed from the tuple because they cannot appear in a frequent itemset. In the rest of this paper, we will consider \( S_{min} = 2 \). The result of this reduction on transactional dataset \( TD_{v0} \) is given by Table II where the number after (:) represents the frequency of an item.

Second, once a tuple is sorted and possibly reduced, it is inserted in the FP-Tree which is essentially a tree where each branch represents the frequent part of a tuple and each node contains the number of branches which share that node (details of the construction of an FP-Tree can be found in [8]). Each edge from a parent to its child is labeled with a value. The root node does not have any label. Figure 1 represents the FP-tree obtained on our running example. Here, we give the modified FP-tree obtained by our approach (see Section V for more details). In the original FP-tree, each node contains the number of transactions containing the itemset represented by the path from the root to the node in question.

Third, nodes that save less values than their parents are removed from the tree. The savings of an itemset \( u \) is \( |u| \ast (f - 1) \) values, where \( f \) is the frequency of \( u \). The remaining itemsets are those relevant for compression.

Finally, FPTCM creates an entry for each frequent itemset \( u \) by scanning the relation to find the tuples that contain \( u \), removes \( u \) from them and adds the reduced tuples to the corresponding sub-table. Tuples that do not contain any frequent itemset are added to a default table. Algorithm 1 summarizes the steps of FPTCM.

B. Weaknesses of FPTCM

FPTCM computes the number of items to save by compressing \( f \) tuples using the frequent itemset \( u \), and compares it to the number of items to save by compressing \( f' \) tuples using the frequent itemset \( u' \) where \( u' \) is the parent of \( u \) (\( f' \geq f \)). If the savings of \( u \) is greater than the savings of \( u' \), then \( u \) is considered as relevant for compression. However, if the number of items to save using \( u \) is fewer compared to the number of items to save using \( u' \), this does not mean that \( u \) is not relevant for compression because the two savings are computed using different number of transactions, leading to a less accurate approximation of the gain of compression.

Let us consider the FP-tree of Figure 1. We can see that itemset \( u = \{v_4 = 1, v_5 = 1\} \) can save 36 items. Consider again the following four itemsets \( u_0 = \{v_4 = 1, v_5 = 1, v_2 = 1\} \), \( u_1 = \{v_4 = 1, v_5 = 1, v_2 = 2\} \), \( u_2 = \{v_4 = 1, v_5 = 1, v_3 = 1\} \) and \( u_3 = \{v_4 = 1, v_5 = 1, v_0 = 1\} \). \( u_0 \) has a frequency equal to 11 and can save 30 items. \( u_1 \) has a frequency equal to 2 and can save 3 items. \( u_2 \) has a frequency equal to 2 and can save 3 items. \( u_3 \) has a frequency equal to 4 and can save 9 items. As itemset \( u' \) is a parent node and can save more items than each of its specialization (i.e., itemsets \( u_0, u_1, u_2 \) and \( u_3 \), FPTCM will consider \( u' \) as relevant for compression. However, considering together itemsets \( u_0, u_1, u_2 \) and \( u_3 \) for compressing tuples can save 45 items.

To create the compressed relation, FPTCM scans the relation and for each tuple \( t_i \), it browses the selected frequent itemsets to find which frequent itemset contains the tuple \( t_i \), then adds the rest of the tuple to the corresponding sub-table. By doing so, FPTCM may require lot of time to complete the compression.

V. FREQUENT PATTERN TREE COMPRESSION METHOD+

(FPTCM+)

In this section, we present our compression method, called FPTCM+, which improves FPTCM in two ways:

- FPTCM+ uses a header table to store single items and their cover. Moreover, each node of the FP-Tree computed by FPTCM+ contains the cover of the itemset represented by the path from the root to that node.

| \( t_0 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 3 |
| \( t_1 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_2 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_3 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_4 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_5 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_6 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_7 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_8 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |
| \( t_9 \) | \( v_4=1 \), 20 | \( v_5=1 \), 19 | \( v_2=1 \), 12 | \( v_3=1 \), 11 | \( v_0=1 \), 10 | \( v_1=2 \), 5 |

**TABLE II:** Tuples sorted according to decreasing frequencies.
Algorithm 1: FPTCM ($TD_c$ : table, $S_{\text{min}}$: integer)

Data: $LFItemsets$: set of itemsets
Result: $R^c_i$: Compressed relation

1. create the header table corresponding to $TD_c$;
2. foreach $t_j \in TD_c$ do
3. sort $t_j$ by decreasing order of value frequency of items and remove values less frequent than $S_{\text{min}}$;
4. add $t_j$ to the FP-Tree $T$; // the nodes represent the frequencies of items
5. end
6. Reduce $T$ by removing nodes with frequency less than $S_{\text{min}}$;
7. foreach child node $n$ of the root node of $T$ do
8. MFItemset($n$, $S_{\text{min}}$, $LFItemsets$);
9. end
10. foreach itemset $\in LFItemsets$ do
11. compress the tuples that contain itemset, end
12. end

This greatly improves the construction of the compressed relation, since the cover of each frequent itemset is known. Consequently, FPTCM+ does not need to verify for each tuple which frequent itemset is contained in.

- Instead of computing the savings for a frequent itemset $u$, we propose in FPTCM+ to consider the compression rate metric (see Definition 3). As for FPTCM, we compare the estimate compression rate with the itemset $u$ against the estimate compression rates with specializations (i.e. super-itemsets) of $u$. If the estimate compression rate with itemset $u$ is lower than the one obtained by each node $(|u| \times (f(u) - 1))$ is less than the one obtained by its parent;

Algorithm 2: FPTCM+ ($TD_c$ : table, $S_{\text{min}}$: integer)

Data: $LFItemsets$: set of itemsets
Result: $R^c_i$: Compressed relation

1. create the header table corresponding to $TD_c$;
2. foreach $t_j \in TD_c$ do
3. sort $t_j$ by decreasing order of value frequency of items and remove values less frequent than $S_{\text{min}}$;
4. add $t_j$ to the FP-Tree $T$; // the nodes represent the covers of itemsets, instead of the frequencies
5. end
6. reduce $T$ by removing nodes with frequency less than $S_{\text{min}}$;
7. foreach child node $n$ of the root node of $T$ do
8. MFItemset($n$, $S_{\text{min}}$, $LFItemsets$);
9. end
10. foreach itemset $\in LFItemsets$ do
11. compress the tuples that contain itemset, end
12. end

A. FPTCM+ in nutshell

In this section, we present the basic steps of our method (see Algorithm 2). The difference between FPTCM+ and FPTCM are highlighted in bold.

First, a header table associated with the transactional dataset $TD_c$ of the constraint relation $R_c$ is created. It is represented as a matrix in which single items and their covers are stored. This step requires one scan of the table. Table III shows the header table associated with the dataset $TD_{c0}$ of Table II and corresponding to the relation $R(c_0)$ of our running example. Then, in the second scan, each transaction is scanned, the set of frequent items in it is sorted in decreasing order of frequency of values and then inserted into the FP-tree $T$ as a branch (see lines 2-3, Algorithm 2). If an itemset shares a

2 A frequent super-itemset of $u$ is obtained by adding to $u$ a frequent item $e_i$ such that $e_i \notin u$ and $e_i$ is the label of the edge from $u$ to its child in the FP-Tree.
Algorithm 3: MFItemset (n: node, S\textsubscript{min}: integer, LFItemsets: set of itemsets)

Result: LFItemsets: set of itemsets

1 if n is a leaf node then
2   Add u to LFItemsets;
3 else
4   add to child\textsubscript{n} the child nodes of n with a
5   frequency greater than or equal to S\textsubscript{min}:
6     sum\textsubscript{f} ← \sum_{v \in \text{child}_n} f(v);
7     compute the compression rate value Rate using
8     itemset u;
9     compute the compression rate value ChRate using
10    the set of frequent super-itemsets of u;
11    if (ChRate < Rate) then
12       Add the itemset u to LFItemsets ;
13    else
14       foreach v ∈ child\textsubscript{n} do
15         MFItemset (v, S\textsubscript{min}, LFItemsets);
16       end
17       if sum\textsubscript{f} < f then
18          Add the itemset u to LFItemsets ;
19       end
20 end

prefix with an itemset already in the tree, this part of the branch
will be shared. In addition, each node in the tree stores the
identifiers of transactions containing the itemset represented
by the path from the root to the node in question. Figure 1
shows the FP-tree obtained on the dataset of Table II. In the
end, nodes with a frequency below the threshold S\textsubscript{min} are
removed from T, because they cannot be frequent itemsets
(line 6, Algorithm 2). In Figure 1, nodes in bold are those
having a frequency greater than or equal to S\textsubscript{min} = 2.

Now, the constructed FP-tree contains all frequency informa-
tion of the dataset. Algorithm 3 summarizes the different
steps to identify in the FP-tree itemsets that are relevant for
compression using the compression rate metric. First, for each
node n that is not a leaf node, we store in a list Child\textsubscript{n} (let
nbrCh be its length) all its children having a frequency (the
frequency is given by the node itself) greater than or equal to
S\textsubscript{min} (line 4 of Algorithm 3). Let u be the frequent itemset

corresponding to the branch from the root to node n and f its
frequency. Second, the sum of frequencies of these children iscomputed (let sum\textsubscript{f} be this sum) (line 5 of Algorithm 3).

To decide which itemset is relevant for compression, we
compare the compression rate values between itemset u (denoted Rate) and its super-itemsets (denoted ChRate) (lines
6-8 of Algorithm 3). If the value of ChRate is below the
value of Rate, then the current itemset u is considered as
relevant for compression, else we consider its super-itemsets
(loop of lines 8-17). For each super-itemset ch, a recursive
call to procedure MFItemset is performed. In line 14, all
transactions that are not covered by the super-itemsets of u
(i.e., sum\textsubscript{f} < f) will be compressed using u (line 15 of
Algorithm 3). In the case where the node n is a leaf one, the
associated itemset is considered as relevant for compression
(lines 1-2 of Algorithm 3).

To complete the compression, we create an entry (u, St)
for each frequent itemset u we have identified. For each
transaction t\textsubscript{i} in cover(u), we remove u from t\textsubscript{i} and we add
it to its corresponding sub-table St. Example 4 details the
different steps of Algorithm 3 on our running example.

B. Summarization using compression rate

To identify itemsets in the FP-tree that are relevant for
compression, we propose to use the compression rate metric.
Let u be the frequent itemset corresponding to the path from
the root note to current node and arity be the arity of the
constraint relation to compress. We have two cases:

1) If the compression is achieved using itemset u, the size
size\textsubscript{a} of the transactions after their compression is equal
to the length of u plus the size of its corresponding
sub-table. The size of the sub-table is obtained by
multiplying the arity of the sub-table (arity — |u|) by
the frequency of u: size\textsubscript{a} = |u| + (arity — |u|) \* f. The size
size\textsubscript{b} of the transactions before their compression
is size\textsubscript{b} = arity \* f. By applying Definition 3, we have

\[
Rate = 1 - \frac{|u| + (arity - |u|) \* f}{arity \* f}
\]

2) if the compression is achieved using the set of frequent
super-itemsets of u, the size size\textsubscript{a} of the transactions
after their compression is defined as follows: size\textsubscript{a} =
size\textsubscript{sup} + size\textsubscript{sub} + size\textsubscript{temp}, where

- size\textsubscript{sup} is the size of super-itemsets of u used for
  compression: size\textsubscript{sup} = nbrCh \* (|u| + 1);
- size\textsubscript{sub} is the size of all the sub-tables obtained
  after compression by super-itemsets of u: size\textsubscript{sub} =
  (arity — (|u| + 1) \* sum\textsubscript{f};
- size\textsubscript{temp} corresponds to the size after compression
  of all transactions not covered by any super-itemset
  of u: size\textsubscript{temp} = (f — sum\textsubscript{f}) \* |u| + |u|.

By applying Definition 3, we have

\[
ChRate = 1 - \frac{size\textsubscript{sup} + size\textsubscript{sub} + size\textsubscript{temp}}{arity \* f}
\] (2)

Using the compression rate metric allows us to evaluate
accurately the real gain that can offer the compression of f
transactions using itemset u against the gain offered by
considering super-itemsets of u. Indeed, we consider exactly
the same number of transactions in the two cases. In contrast,
FPTCM compares the savings obtained when using both the
frequent itemset u and the parent of u. However, both cases
use different sets of transactions to compute this savings.
This gives a less accurate result in terms of compression gain.
Example 4: In our example, the node corresponding to itemset \((v_4 = 1)\) will be explored first. Only the child node \((v_5 = 1)\) has a frequency greater than \(S_{min} = 2\). Therefore, super-itemset \((v_4 = 1, v_5 = 1)\) will be considered (\(nbrCh = 1\) and \(sum_f = 19\)). For itemset \((v_4 = 1)\), the value of \(Rate\) is equal to 0.15, while for itemset \((v_4 = 1, v_5 = 1)\), \(ChRate = 0.3\). As \((ChRate > Rate)\), the child node \((v_5 = 1)\) becomes the current node. It has four child nodes (\(nbrCh = 4\)) and the sum of their frequencies \(sum_f\) is equal to 19. The value of \(Rate\) for itemset \((v_4 = 1, v_5 = 1)\) is equal to 0.31, while the value of \(ChRate\) for its four super-itemsets is equal to 0.39. As again \((ChRate > Rate)\), the child nodes of the node \((v_5 = 1)\) will be considered. We start by the child \((v_0 = 1)\); all its child nodes are not frequent. So, transactions stored in the node \((v_0 = 1)\) (i.e., \(t_4, t_8, t_9\) and \(t_{10}\)) will be compressed using the frequent itemset \((v_0 = 1, v_4 = 1, v_5 = 1)\). When the algorithm completes the exploration of all the nodes of the FP-tree, we get the following set of frequent itemsets used for the compression and the associated compressed transactions:

- \((v_1 = 1, v_2 = 1, v_4 = 1, v_5 = 1)\), \(\{t_{11}, t_{12}, t_{16}, t_{17}, t_{18}\}\).
- \((v_0 = 1, v_2 = 1, v_4 = 1, v_5 = 1)\), \(\{t_0, t_1, t_2, t_3, t_5, t_6\}\).
- \((v_0 = 1, v_4 = 1, v_5 = 1)\), \(\{t_4, t_8, t_9, t_{10}\}\).
- \((v_3 = 1, v_2 = 1, v_4 = 1, v_5 = 1)\), \(\{t_{13}, t_{19}\}\).
- \((v_1 = 4, v_2 = 1, v_4 = 1, v_5 = 1)\), \(\{t_{14}, t_{15}\}\).

Table IV shows the different entries corresponding to the five frequent itemsets.

To evaluate the savings in terms of items that can be obtained by our approach on the running example, we evaluate the size of the constraint relation before and after the compression. Let \(arity = 6\) be the arity of \(R(c_0)\) and \(nbrTup = 21\) be the number of tuples in \(R(c_0)\). The size of \(R(c_0)\), denoted by \(size(R(c_0))\), is \(size(R(c_0)) = arity \times nbrTup = 126\). The size of the compressed relation is equal to 86 items. So, the savings that can be obtained is 40 values.

When considering \(FPTCM\) approach, two itemsets are used for compressing the constraint relation \(R(c_0)\):

- \((v_1 = 4, v_2 = 2, v_3 = 1, v_4 = 1, v_5 = 1)\) that compresses two transactions \(t_{13}, t_{19}\).
- \((v_2 = 1, v_4 = 1, v_5 = 1)\) that compresses the set of transactions \(\{t_0, t_1, t_2, t_3, t_5, t_6, t_{11}, t_{12}, t_{16}, t_{17}, t_{18}\}\).

The size of the relation \(R(c_0)\) after compressing it using the two frequent itemsets is equal to 94 items. So, we can see that \(FPTCM^+\) provides a good compression compared to \(FPTCM\).

C. Solving the compressed constraint relations

The structure of the compressed relations is exactly the same as the one proposed in [1]. So, to solve compressed CSP, we used in this work, the MaxDeg heuristic [6] to sort the variables of the CSP and the Graph-based backjumping algorithm [7] (GBJ) to fix the variable choice during the resolution process. Note that the difference between the two methods (\(FPTCM\) and \(FPTCM^+\)) is just at the compression step.

VI. EXPERIMENTS

In this section we present an experimental evaluation of our method. The experiments have been performed on Intel Core i5 2.5 GHz. We compare our approach \(FPTCM^+\) with \(FPTCM\) method on the same benchmarks\(^3\) used in [1]. Table VI shows the characteristics of these benchmarks. For each benchmark, we give the number of instances that it contains (\(nbrInst\), the maximum number of variables (\(V\)), the greatest value of domains (\(|D|\)), the largest number of relations (\(nbrR\)), the size of the largest constraint relation (\(R_{max}\)), the largest constraint arity (\(arity\)), the greatest number of constraints (\(nbrC\)). All methods have been implemented in Java and use benchmarks

\(^3\)Datasets available at https://www.cril.univ-artois.fr/~lecoutre/#/
<table>
<thead>
<tr>
<th>Datasets</th>
<th>Modified</th>
<th>Renault</th>
<th>ukVg</th>
<th>ogdVg</th>
<th>Rand10-20-10</th>
<th>wordVg</th>
</tr>
</thead>
<tbody>
<tr>
<td>T&lt;sub&gt;comp&lt;/sub&gt;</td>
<td>FPTCM&lt;sup&gt;+&lt;/sup&gt;</td>
<td>44.70</td>
<td>2.2</td>
<td>23.85</td>
<td>17.65</td>
<td>20.20</td>
</tr>
<tr>
<td>FPTCM&lt;sup&gt;-&lt;/sup&gt;</td>
<td>44.70</td>
<td>2.2</td>
<td>19.65</td>
<td>17.65</td>
<td>20.20</td>
<td>33.65</td>
</tr>
<tr>
<td>T&lt;sub&gt;rs&lt;/sub&gt;</td>
<td>FPTCM&lt;sup&gt;+&lt;/sup&gt;</td>
<td>6.8</td>
<td>5.28</td>
<td>2.28</td>
<td>13.37</td>
<td>5.2</td>
</tr>
<tr>
<td>FPTCM&lt;sup&gt;-&lt;/sup&gt;</td>
<td>6.7</td>
<td>8.62</td>
<td>11.41</td>
<td>19.7</td>
<td>9.35</td>
<td>1.09</td>
</tr>
<tr>
<td>T&lt;sub&gt;tot&lt;/sub&gt;</td>
<td>FPTCM&lt;sup&gt;+&lt;/sup&gt;</td>
<td>217.52</td>
<td>0.68</td>
<td>146.71</td>
<td>52.41</td>
<td>3.4</td>
</tr>
<tr>
<td>FPTCM&lt;sup&gt;-&lt;/sup&gt;</td>
<td>217.58</td>
<td>1.21</td>
<td>187.04</td>
<td>6.2</td>
<td>6.48</td>
<td>46.96</td>
</tr>
<tr>
<td>Rate</td>
<td>FPTCM&lt;sup&gt;+&lt;/sup&gt;</td>
<td>79.18</td>
<td>80.08</td>
<td>28.24</td>
<td>38.45</td>
<td>18.93</td>
</tr>
<tr>
<td>FPTCM&lt;sup&gt;-&lt;/sup&gt;</td>
<td>79.06</td>
<td>37.09</td>
<td>21.97</td>
<td>23.7</td>
<td>18.09</td>
<td>18.84</td>
</tr>
</tbody>
</table>

TABLE V: Comparing FPTCM<sup>+</sup> FPTCM in terms of average compression rate and resolution time. Line <i>ins<sub>rs</sub></i> gives the number of instances solved within the time limit of one hour, line <i>T<sub>comp</sub></i> shows the average CPU time (in second) to compress an instance, line <i>T<sub>rs</sub></i> reports the average CPU time for succeeded instances, line <i>T<sub>tot</sub></i> shows the average global CPU time for the two steps, and line <i>Rate</i> gives the average compression rate (in percentage).

The same resolution approach for solving the compressed CSP. For our experiments, we fixed <i>S<sub>min</sub></i> to 2 and considered for compression only relations with at least 10 tuples and with arity greater than 2. Table V compares the performance of the two methods. For each method and each dataset, we report the number of instances solved within the time limit of 1h (<i>ins<sub>rs</sub></i>), the average CPU time (in second) to compress an instance (<i>T<sub>comp</sub></i>), the average CPU time for succeeded instances (<i>T<sub>rs</sub></i>), the average global CPU time for the two steps (<i>T<sub>tot</sub> = T<sub>comp</sub> + T<sub>rs</sub></i>), and the average compression rate (in percentage) obtained (<i>Rate</i>). The average compression rate for an instance is the size of relations after their compression over the size of the original relations. For each dataset, the average compression rate is computed by dividing the sum of compression rates obtained for each instance by the total number of compressed instances.

As we can see, both methods perform similarly in terms of the number of instances solved except on crossword-uk-vg dataset where FPTCM<sup>+</sup> succeeded to solve 4 more instances than FPTCM. When comparing the average compression rate, we can observe that FPTCM<sup>+</sup> performs better, particularly on the two instances Modified – Renault and Renault where the gain in terms of compression rate is greatly amplified; it reaches on average 80% against about 37% for FPTCM. Table V also shows that FPTCM<sup>+</sup> takes less time than FPTCM<sup>-</sup>.

An instance is succeeded if it can be solved within the time limit.

![Fig. 2: The average total time required by FPTCM and FPTCM<sup>+</sup> for solving each dataset.](image)

STACK TABLE VII: Comparing FPTCM with FPTCM<sup>+</sup>: frequent itemsets vs. entries.

![TABLE VI: Characteristics of datasets.](image)
VII. CONCLUSION

In this paper, we have proposed a new approach based on data mining techniques for compressing constraint relations. Our approach improves the one proposed by Gharbi et al. [1] in two ways: (i) the use of the cover of an itemset in the FP-tree instead of its frequency, thus enhancing the construction of entries since the cover of each frequent itemset is known, (ii) the use of compression rate metric instead of savings to compute frequent itemsets relevant for compression, thus allowing higher compression and efficient resolution of compressed tables by identifying more accurate frequent itemsets. We have implemented the \textit{MaxDeg} heuristic and the backjumping algorithm to solve the compressed CSPs and we evaluated our approach on some benchmarks. Experiment results showed that our approach (called \textit{FPTCM}) offers better compression rates compared to the approach of [1]. Moreover, our approach obtains better CPU times for both compression and solving steps.

REFERENCES