Inverse Tree-OLAP: Definition, Complexity and First Solution

Domenico Saccà[1], Edoardo Serra[2], and Alfredo Cuzzocrea[3]

[1] University of Calabria, Italy
sacca@unical.it
[2] Boise State University, USA
edoardoserra@boisestate.edu
[3] University of Trieste and ICAR-CNR, Italy
alfredo.cuzzocrea@dia.units.it

Abstract. Count constraint is a data dependency that requires the results of given count operations on a relation to be within a certain range. By means of count constraints a new decisional problem, called the Inverse OLAP, has been recently introduced: given a flat fact table, does there exist an instance satisfying a set of given count constraints? This paper focus on a special case of Inverse OLAP, called Inverse Tree-OLAP, for which the flat fact table key is modeled by a Dimensional Fact Model (DFM) with a tree structure.

1 Introduction

Emerging "Big Data" platforms and applications call for the invention of novel data analysis techniques that are capable to handle large amount of data. There is therefore an increasing need to use real-life datasets for data-driven experiments but, as pointed out in a recent ACM SIGMOD Blog post by Gerhard Weikum [13], datasets used into research papers are often poor. Companies have their own interesting data, and industrial labs have access to such data and real-life workloads; however, such datasets are often proprietary and out of reach for academic research. In order to ensure high quality experimental findings, inverse mining techniques can be applied to generate artificial datasets that reflect the patterns of real ones: the patterns are first discovered by data mining techniques (or directly provided by domain experts) and then used to generate "realistic" privacy-preserving datasets.

A promising and important example of inverse mining is Inverse Frequent itemset Mining (IFM), first stated in [9], which consists of finding a transactional database $\mathcal{D}$ satisfying given support constraints on some itemsets, which are typically the frequent ones. As an example, given the items a, b, c, d, the itemsets $I_1 = \{a, b\}$, $I_2 = \{b, c\}$ and $I_3 = \{c, d\}$, the support constraints $\sigma_{I_1} = \sigma_{I_2} = 100$ and $\sigma_{I_3} = 50$, and a fixed database size equal to 170, a feasible transactional database $\mathcal{D}$ consists of the following (replicated) transactions: $\langle\{a, b, c\}, 70\rangle$ (i.e., there are 70 occurrences of the transaction $\{a, b, c\}$ in $\mathcal{D}$), $\langle\{b, c, d\}, 10\rangle$, $\langle\{a, b\}, 30\rangle$, $\langle\{b, c\}, 20\rangle$ and $\langle\{c, d\}, 40\rangle$. Observe that the support constraint $\sigma_{I_1} = 100$ is satisfied by $\mathcal{D}$ as there are 30 occurrences of I_1 in

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

$\mathcal{D}$ and I_1 is a subset of $\{a, b, c\}$ occurring 70 times in $\mathcal{D}$ – it is easy to see that the other support constraints are satisfied as well.

Count constraints have been used in [12] to define a new integrity constraint decisional problem: given a relation scheme and a number of count constraints on it, does there exist a relation instance satisfying the count constraints? The problem has been called *Inverse OLAP* because the typical relation scheme dealt with is a (*flat*) *fact table*, whose attributes are dimensions (i.e., properties, possibly structured at various levels of abstraction) – no measure attributes are considered as the only aggregation operator is the count. We recall that a flat fact table can be represented as a *star schema* with additional tables for storing dimensions or a *snowflake schema*, which includes additional dimensional tables describing dimension hierarchies. This kind of fact table is called a *factless fact table* [7] and it arises in some scenarios storing many-to-many mappings in which no attribute qualifies as a measure – typical cases are event records, where an event is given by a combination of simultaneously occurring dimensional characteristics.

Inverse OLAP has a potential high relevance in synthesizing data cubes having predefined characteristics to be used in benchmarks of novel techniques for handling big data. Inverse OLAP has been proved in [12] to be NEXP-complete under various conditions: data complexity (i.e, the number of attributes and the size of constraints are constant), program complexity (i.e, the domains are constant) and combined complexity.

We are not discouraged by the high complexity and in this paper we present a solution approach of the Inverse OLAP problem for the case of a flat fact table, called *tree-fact table*, whose schema is a special case of Dimensional Fact Model (DFM) of [5]: it is a tree (and not a graph as in the general definition) having dimension hierarchy levels as nodes and the "rolls-up-t" relationships between them as edges.

This paper is the short version of the paper [11], where we present the main results of our research.

2 Inverse Tree-OLAP

In this section we define the *Inverse Tree-OLAP* problem. We assume that the flat fact table $\mathcal{R}$ has the following scheme (*tree-fact table*): $\mathcal{R}(B_1, \ldots, B_n, S^1, \ldots, S^n, M_1, \ldots, M_m)$, where $(B_1, \ldots, B_n)$ is the table key (*basic dimensions*), S^i, $1 \leq i \leq n$, is a dimension hierarchy tree rooted on B_i, and $M_1, \ldots, M_m)$ are measures. We also assume that the all the once-to-many relationships in the subtrees $S^2, \ldots, S^n$ are stored as facts in ad-hoc additional domain tables (as in an ontology) whereas the first level of the one-to-many relationships in the subtree S^1 are not. In addition we assume that all the dimension domains are given in suitable tables except for the basic dimension B_1 (*aggregating basic dimension*), whose values are generated by means of a polynomial time function. Finally the values of measures are determined by means of a polynomial time function having the table key domains as range.

We make the following restrictions on the count constraints that can be applied to a tree-fact table:

- *Tree-Tuple Count Constraint*: the table key $\{B_1, \ldots, B_n\}$ must be contained in $\mathbf{X} \cup \mathbf{Y}$, being $\mathbf{X}$ and $\mathbf{Y}$ suitable subsets of dimensions, respectively;
- *Tree-Tuple Count Constraint-B_1*: $\mathbf{Y} = \{B_1\}$, $\mathbf{X}$ is a subset of dimensions in the subtree S^1 rooted at A_1 and all other basic dimensions $B_2, \ldots, B_n$ as well as the ones in the corresponding subtrees $S^2, \ldots, S^n$ must be contained in $\mathbf{Z}$, being $\mathbf{Z}$ a suitable subset of dimensions;
- *Tree-Group Count Constraint*: $\mathbf{W} = B_1$, $\mathbf{Y} = \{B_2, \ldots, B_n\}$ while all other dimensions must be in $\mathbf{Z}$ (being $\mathbf{Z}$ a suitable subset of dimensions), i.e., they are existentially quantified.

Observe that Tree-Group Count Constraints define an IFM-like subproblem, whose very objective is to model how the values of the basic dimensions can occur together in a many-to-many relationship. Restrictions on Tree-Tuple Count Constraints avoid the problem of handling duplicates on table projections. In fact, the projected tuples are guaranteed to be distinct as the table key $B_1, \ldots, B_n$ is included in the set of selection / projection attributes. The problem of removing duplicates from table projections has been recognized in [1] as the critical step for checking satisfaction of cardinality constraints. As for Tree-Tuple Count Constraint-B_1, distinctness of projected tuples will be guaranteed by our solution approach that group the tuples by the aggregation basic dimension.

We are now ready to define the specialization of Inverse OLAP.

Inverse Tree-OLAP Problem. Given a tree-fact table $\mathcal{R}$ on a DFM D and a set of 2D count constraints C, the *Inverse Tree-OLAP* problem consists of deciding whether there exists a relation r on $\mathcal{R}$ such that $r \models C \cup F$, where F are the functional dependencies corresponding to D. □

The next result shows that the (both data and combined) complexity of Inverse Tree-OLAP remains NEXP-complete.

Proposition 1. *Inverse Tree-OLAP is* NEXP*-complete.*

PROOF (*sketch*). Membership to NEXP derives from the fact that Inverse OLAP is in NEXP and Inverse Tree-OLAP is a specialization of Inverse OLAP. Actually satisfaction of FDs is not explicitly required in the general Inverse OLAP formulation as possible FDs are expressed as count constraints. Checking FDs can be obviously done in time polynomial in the size of a problem certificate and, therefore, the complexity remains in NEXP. To prove NEXP-hardness, it is sufficient to show that any instance x of $\mathtt{IFM}_\mathtt{I}$ can be transformed in logarithmic space into an instance x' of Inverse Tree-OLAP, such that x is a YES-instance for $\mathtt{IFM}_\mathtt{I}$ if and only if x' is a YES-instance for Inverse Tree-OLAP. We shall give some intuition on this part of the proof in the example below. □

Observe that NEXP-completeness only depends on the presence of Tree-Group Count Constraints. The fact that Tuple Constraints do not have high complexity is due to our restrictions that produce table projections without duplicates.

To get an intuition of our approach, consider a tree-fact table $\mathcal{R}(S, W, B, C, R, T, L, A, G, P)$ modeling a classical example of point-of-sales transaction application. Recall that the dimensions are: S (Sale), W (Ware), B (ware Brand), C (sale

Customer), R (customer Ranking), T (customer Town), L (sale Location), A (sale Area) and G (ware Group). The basic dimensions are S, W and B and S is the aggregating one. There is a unique measure: P (item Price), where an item is a pair ware-brand and its price may vary from one location to another. Then the value of P is univocally determined by the values of L, W and B – we assume that the function returning the price value is suitably stored to be used after a feasible solution has been produced. The domains of the dimensions, denoted by $\mathcal{D}_S$, $\mathcal{D}_W$ and so on, are given and they are finite tables; however, $\mathcal{D}_S$ is not explicitly stored but the values are generated as needed.

The prefixed one-to-many relationships are given and stored into the tables: $\mathcal{D}_{WG}$, $\mathcal{D}_{LA}$, and $\mathcal{D}_{CT}$. The other one-to-many relationships are not predefined.

A domain table $\mathcal{D}_{\{W\,B\}\sigma_1\sigma_2}$ stores a number of predefined itemsets (i.e., sets of items) and a range of admissible supports $\sigma_1 \leq \sigma_2$ for each of them. There are additional tables: $\mathcal{D}_{GA\,\alpha_1\alpha_2}$ storing the admissible numbers of ware groups that must be assigned for location areas, $\mathcal{D}_{BR\,\gamma_1\gamma_2}$ storing the admissible numbers of brands that must be sold to all customers of each ranking and $\mathcal{D}_{R\,\omega_1\omega_2}$ storing the admissible numbers of regions that must be interested to all customers of each ranking. Observe that the minimum value ω_1 for the lowest ranking is 0.

Enforcing the total number of sales

$$true \;\rightarrow\; s_1 \leq \#(\{\mathtt{S} : \mathcal{R}(\mathtt{S},\mathtt{w},\mathtt{b},\mathtt{c},\mathtt{r},\mathtt{t},\mathtt{l},\mathtt{a},\mathtt{g},\mathtt{p})\}) \leq s_2 \tag{1}$$

where low letters represent existentially quantified variables and s_1, s_2 with $s_1 \leq s_2$ are two non negative integers, denoting respectively the minimal and the maximal number of sales in a feasible solution. Constraint (1) is a Tree-Tuple Count Constraint-B_1.

Enforcing frequency support constraints on itemsets

$$\begin{aligned}\mathcal{D}_{\{\mathtt{WB}\}\sigma_1\sigma_2}(\ddot{\mathtt{X}}, \ddot{\sigma}_1, \ddot{\sigma}_2) \;\rightarrow\; \\ \ddot{\sigma}_1 \leq \#(\{\mathtt{S} : \ddot{\mathtt{X}} \subseteq \{\mathtt{WB} :\; \mathcal{R}(\mathtt{S},\mathtt{W},\mathtt{B},\mathbf{v})\}\}) \leq \ddot{\sigma}_2\end{aligned} \tag{2}$$

where $\mathbf{v}$ is the list of existentially quantified variables $[\mathtt{c},\mathtt{r},\mathtt{t},\mathtt{l},\mathtt{a},\mathtt{g},\mathtt{p}]$, dotted letters represent universally quantified variables and $\ddot{\sigma_1}$, $\ddot{\sigma_2}$ are the minimal and maximal support for frequent itemsets. Constraint (2) is a Tree-Group Count Constraint that enforces that all itemsets stored in $\mathcal{D}_{\{I\}\beta_1\beta_2}$ be frequent.

Enforcing infrequency support constraints on itemsets

$$\begin{aligned}\ddot{\mathtt{Y}} = &\{W\,B : \mathcal{D}_W(W) \times \mathcal{D}_B(B)\} \wedge \mathcal{D}_{\{W\,B\}\sigma_1\sigma_2}(\ddot{X}, \ddot{\sigma}_1, \ddot{\sigma}_2) \wedge \\ &\ddot{Y} \not\subseteq \ddot{X} \;\rightarrow\; \#(\{\mathtt{S} : \ddot{\mathtt{X}} \subseteq \{\mathtt{WB} :\; \mathcal{R}(\mathtt{S},\mathtt{W},\mathtt{B},\mathbf{v})\}\}) \leq \sigma\end{aligned} \tag{3}$$

where σ is a non negative integer denoting the infrequency support threshold. Constraint 3 is a Tree-Group Count Constraint that enforces that all itemsets that are not a (not necessarily proper) subset of some itemset in $\mathcal{D}_{\{W\,B\}\sigma_1\sigma_2}$ must have a support below the threshold σ.

We point out that Constraints (1), (2) and (3) expresses the $\mathtt{IFM}_\mathtt{I}$ problem – so they can be used in the NEXP-hardness proof of Proposition 1. Our formalism is rather powerful as it enable to define additional constraints to $\mathtt{IFM}_\mathtt{I}$ – as an example, we next show how to enforce a size range for the fact table and a maximum number of items for transactions (the latter constraint has been analyzed in [3]).

Enforcing the total number of tuples in the fact table

$$true \;\rightarrow\; s_3 \leq \#(\{\mathtt{S\,W\,B} : \mathcal{R}(\mathtt{S},\mathtt{W},\mathtt{B},\mathbf{v})\}) \leq s_4 \tag{4}$$

where s_3, s_4 with $s_3 \leq s_4$ are two non negative integers, denoting respectively the minimal and the maximal cardinality of a feasible fact table instance. Constraint (1) is a Tree-Tuple Count Constraint.

Enforcing a maximum number of itemsets in a sale

$$\mathcal{D}_{\mathtt{S}}(\ddot{\mathtt{S}}) \;\rightarrow\; \#(\{\mathtt{W\,B} :\; \mathcal{R}(\ddot{\mathtt{S}},\mathtt{W},\mathtt{B},\mathbf{v})\}) \leq \lambda \tag{5}$$

where λ is a positive integer denoting the maximal transaction length. Constraint 5 is a Tree-Tuple Count Constraint.

We next define a number of additional Tree-Tuple Count Constraints (including type $B-1$) that involve also non-basic dimensions.

Enforcing prefixed one-to-many relationships Consider first the prefixed one-to-many relationship from G (ware Group) to W (Ware), defined by $\mathcal{D}_{W\,G}$. The following Tree-Tuple Count Constraint:

$$\begin{aligned}&\mathcal{D}_W(\ddot{\mathtt{W}}) \wedge \mathcal{D}_{\mathtt{S}}(\ddot{\mathtt{S}}) \wedge \mathcal{D}_{\mathtt{B}}(\ddot{\mathtt{B}}) \;\rightarrow\\ &\quad \#(\{\mathtt{G} :\; \mathcal{R}(\ddot{\mathtt{S}},\ddot{\mathtt{W}},\ddot{\mathtt{B}},\mathtt{c},\mathtt{r},\mathtt{t},\mathtt{l},\mathtt{a},\mathtt{G},\mathtt{p}) \wedge \neg\mathcal{D}_{\mathtt{W\,G}}(\ddot{\mathtt{W}},\mathtt{G})\}) = 0.\end{aligned} \tag{6}$$

excludes the possibility that any ware can take a group value different from the one that has been fixed in $\mathcal{D}_{W\,G}$. Observe that we have slightly modified the notation inside the function # by enabling a conjunction of the fact predicate with domain predicates – the semantics of this extension is obvious.

We add similar constraints for all other prefixed one-to-many relationships: from L (sale Location) to A (Area), from C (Customer) to T (Town) and from C to R (customer Ranking).

Distributing ware groups on areas

$$\begin{aligned}&\mathcal{D}_{GA\,\alpha_1\alpha_2}(\ddot{G},\ddot{A},\ddot{\alpha}_1,\ddot{\alpha}_2) \;\rightarrow\\ &\quad \ddot{\alpha}_1 \leq \#(\{\mathtt{S\,W\,B} :\; \mathcal{R}(\mathtt{S},\mathtt{W},\mathtt{B},\mathtt{c},\mathtt{r},\mathtt{t},\mathtt{l},\ddot{\mathtt{A}},\ddot{\mathtt{G}},\mathtt{p})\}) \leq \ddot{\alpha}_2.\end{aligned} \tag{7}$$

This Tree-Tuple Count Constraint fixes the number of wares of each group that must be sold to all locations of a given area.

Distributing sales on locations

$$\mathcal{D}_{\mathtt{L}}(\ddot{\mathtt{L}}) \;\rightarrow\; \gamma_1 \leq \#(\{\mathtt{S} :\; \mathcal{R}(\mathtt{S},\mathtt{W},\mathtt{B},\mathbf{v})\}) \leq \gamma_2. \tag{8}$$

where $\mathbf{v}$ has been defined above and γ_1, γ_2 with $\gamma_1 \leq \gamma_2$ are two non negative integers denoting the same distribution range for all locations. This is a Tree-Tuple Count Constraint-B_1.

Assigning brands to rankings

$$\begin{aligned}&\mathcal{D}_{\mathtt{B\,R}\,\tau_1\tau_2}(\ddot{B},\ddot{R},\ddot{\tau}_1,\ddot{\tau}_2) \;\rightarrow\\ &\quad \tau_1 \leq \#(\{\mathtt{S\,W}\ddot{\mathtt{B}} :\; \mathcal{R}(\mathtt{S},\mathtt{W},\ddot{\mathtt{B}},\mathtt{c},\ddot{\mathtt{R}},\mathtt{t},\mathtt{l},\mathtt{a},\mathtt{g},\mathtt{p}\}) \leq \tau_2.\end{aligned} \tag{9}$$

This is a Tree-Tuple Count Constraint.

Assigning the numbers of sales to customers per ranking

$$\mathcal{D}_{\mathtt{R}\,\omega_1\omega_2}(\ddot{R}, \ddot{\omega}_1, \ddot{\omega}_2) \;\rightarrow \\ \omega_1 \leq \#(\{\mathtt{S} : \mathcal{R}(\mathtt{S}, \mathtt{w}, \mathtt{b}, \mathtt{c}, \ddot{\mathtt{R}}, \mathtt{t}, \mathtt{l}, \mathtt{a}, \mathtt{g}, \mathtt{p})\}) \leq \omega_2. \tag{10}$$

This is a Tree-Tuple Count Constraint-B_1. Observe that a ranking dimension is considered in [8] to show an example of derived (hidden) dimensions.

3 Solving Inverse Tree-OLAP

In this section we present a method for finding an approximate solution of the Inverse Tree-OLAP. Given a instance x of the Inverse Tree-OLAP:

1. Construct an instance y of $\mathtt{IFM_I}$ using Tree-Group Count Constraints and possible Tree-Tuple Count Constraints (also in the variant B_1) for which all non-basic dimensions are existentially quantified. Any itemset is a set of $(n-1)$-tuples on the basic dimensions $B_2, \ldots, B_n$ that share the same value for the aggregating basic dimension B_1.
2. Solve y using the algorithm of [6] – let D be the transaction database computed by the algorithm, where a transaction is a set of items and each item is an element of $B_2 \times \cdots \times B_n$.
3. Represent the database D as a set of triples $(t, Z(t), \delta(t))$, where $Z(t)$ is an itemset occurring in D, $\delta(t)$ is the number of its duplicates in D and t is an identifier for the itemset $Z(t)$. Note that t is not a value in the domain of the aggregating basic domain B_1 but it represents a group of $\delta(t)$ distinct values for B_1.
4. Represent an instance r of the relation $\mathcal{R}$ by means of variables as follows: for each $(t, Z(t), \delta(t))$, introduce variables $x_{\hat{b}_1 b_2 \ldots b_n \mathbf{d}}$, where $b_2 \ldots b_n\, \mathbf{d}$ are all admissible values for $B_2, \ldots, B_n$, $\mathbf{d}$ is the list of all possible values for non-basic domains that are not determined by the prefixed one-to-many relationships and $\hat{b}_1$ represents the portion of t duplicates that are assigned to the values of the other subscript indices. Observe that this representation can be seen as a *chase*, which is a finite "canonical" database, witnessing the satisfiability of integrity constraints [2].
5. Use the Tree-Tuple Count Constraints and Tree-Tuple Count Constraints-B_1 involving non-basic dimensions to compute the variable values. To this end, we use linear equations that are solved using a linear program whose objective function is the cost of violating the above constraints.
6. Generate a flat table starting from the computed variable values by including the values for non-basic domains that are determined by the prefixed one-to-many relationships and by computing the measure values.

For presentation sake, we next illustrate our method by referring to the example described in Section 2.

As already mentioned, constraints (1), (2) and (3) encode an instance of $\mathtt{IFM_I}$, described in the following. We first present the formal definition of $\mathtt{IFM_I}$. Let:

1. $\mathcal{I}$ be a given set of n items and $\mathcal{U}_\mathcal{I}$ be the set of all non-empty itemsets over $\mathcal{I}$
2. S be a given set of m itemsets over the items in $\mathcal{I}$

3. S' denote $\{I \in \mathcal{U}_{\mathcal{I}} \mid I' \in S : I \subseteq I'\}$
4. $\Gamma_\sigma = \{(I, \sigma_1^I, \sigma_2^I) \mid I \in S,\ 0 \leq \sigma_1^I \leq \sigma_2^I\}$ be a given set of triples assigning a minimum and maximum support to each itemset in S
5. $\sigma' \geq 0$ be the maximum support threshold for all itemsets in S'
6. $\mathbf{size} = (size_1, size_2)$, $0 \leq size_1 \leq size_2$, be the minimal and maximal number of transactions.

The *inverse frequent itemset mining problem with infrequency constraint* ($\texttt{IFM}_\texttt{I}$) on $\mathcal{I}$, S, Γ_σ, σ' and **size** consists of finding a database D over $\mathcal{I}$ such that the following conditions hold (or of eventually stating that there is no such a database):

$$\forall I \in S : \sigma_{min}^I \leq \sigma^D(I) \leq \sigma_{max}^I \tag{11}$$

$$\forall I \in S' : \sigma^D(I) \leq \sigma' \tag{12}$$

$$size_1 \leq |D| \leq size_2. \tag{13}$$

By the anti-monotonocity property, constraints (12) can be replaced by:

$$\forall I \in B_{S'} : \sigma^D(I) \leq \sigma' \tag{14}$$

where $B_{S'} = \{I \in S' \mid I' \in S' : I' \subset I\}$ is the set of the bottom (i.e., minimal) elements of S'.

An instance of $\texttt{IFM}_\texttt{I}$ is constructed by fixing: $\mathcal{I} = \{\langle \texttt{w b} \rangle \mid \mathcal{D}_\texttt{W}(\texttt{w}) \times \mathcal{D}_\texttt{B}\texttt{b}\}$, $\Gamma_\sigma = \{(\texttt{X}, \sigma_1, \sigma_2) \mid \mathcal{D}_{\{\texttt{WB}\}\sigma_1\sigma_2}(\texttt{X}, \sigma_1, \sigma_2)\}$, $\sigma' = \sigma$ (see constraint 3), $size_1 = s_1$ and $size_2 = s_2$ (see constraint 1). It is then straightforward to derive S and $B_{S'}$.

For the solution of the $\texttt{IFM}_\texttt{I}$ instance, we adopt the approach of [6]. The instance is represented as an integer linear program with a linear number of constraints (corresponding to the support constraints of the itemsets in $S \cup B_{S'}$ and to the two database size constraints) and an exponential number 2^n of variables x_j, one for each possible transaction I_j, indicating to number of its occurrences in the database. The program is represented in a succinct format with size $\mathcal{O}(n\,(m + m'))$ – recall that a succinct problem is a problem whose instances are not given straightforward, but are themselves encoded using an a concise input with size logarithmic in the actual input size [10]. Then the integer constraint on the variables x_j are relaxed so that an approximate solution is obtained by solving a linear program. Because of the exponential number of variables, the *column generation* algorithm (see e.g [4]), which is a version of the simplex dealing with a large number of variables (large-scale linear programs). This method solves a linear program without explicitly including all columns (i.e., variables), in the coefficient matrix but only a subset of them with cardinality equal to the number of rows (i.e., constraints). Columns are dynamically generated by solving an auxiliary optimization problem called the *pricing problem*.

Observe that constraints (4) and (5) are not taken into account in the resolution algorithm of [6]. Constraint (4) can be easily handled by a simple additional constraint in the liar program. Constraint (5) states that for each itemset with length greater than λ, the associated variable x_j must be equal to zero. This restriction can be easily handled inside the pricing problem resolution.

The algorithm returns a transaction database D that is an approximate solution of the problem. The database D is represented in a succinct format: $(\hat{s}, Z, \delta)$, where Z is

an itemset occurring in D, δ is the number of its duplicates and $\hat{s}$ is an identifier for Z. We stress that $\hat{s}$ is not a sale value in $\mathcal{D}_S$ as the itemset Z will be eventually assigned to δ distinct sale values. Let $\hat{S}$ the set of all identifiers $\hat{s}$. For each $\hat{s} \in \hat{S}$, $\delta(\hat{s})$ denotes the duplicate number for $\hat{s}$, $Z(\hat{s})$ is the itemset Z.

So far we have implemented steps (1)-(3). To implement step (4), we introduce the variables $x_{\hat{s}wblc}$ for each $\hat{s}$ in $\hat{S}$, for each ware w of the domain of W, for each brand b of the domain of B, for each location l of the domain L and for each customer c of the domain of C. There are no indices corresponding to non-basic domains that are defined by means of on-to-many fixed relationships. Indeed, because of constraints of type (6), once that a domain value is fixed, the value of its roll-up domain value is automatically fixed as well.

To make the computation effective, we relax the integer constraint on all variables; so they are defined on the domain of non-negative rational numbers. We introduce the following equations:

$$\sum_{l \in \mathcal{D}_L, c \in \mathcal{D}_C} x_{\hat{s}wblc} = \delta(\hat{s}) \qquad \forall \hat{s} \in \hat{S}\ \forall \langle w, b \rangle \in Z(\hat{s}) \tag{15}$$

$$x_{\hat{s}wblc} = 0 \qquad \begin{array}{c} \forall \hat{s} \in \hat{S}\ \forall l \in \mathcal{D}_L\ \forall c \in \mathcal{D}_C \\ \forall \langle w, b \rangle \in (\mathcal{D}_W \times \mathcal{D}_B) \setminus Z(\hat{s}) \end{array} \tag{16}$$

Equation (15) states that, fixed a transaction $\hat{z}$ and any item $\langle w, b \rangle$ of $Z(\hat{S})$, the sum of all assignments of the item to all possible locations and customers must be equal to the duplicate number $\delta(\hat{s})$. On the other hand, equation 16 enforces that $x_{\hat{s}wblc} = 0$ whenever the item $\langle w, b \rangle$ i is not part of any transaction $\hat{s}$.

Let us now implement step (5). To this end, we consider all constraints from (7) to (10).

Constraints (7) is implemented as:

$$\alpha_1^{ga} \leq \sum_{\substack{\forall \hat{s} \in \hat{S}\ \forall c \in \mathcal{D}_C\ \forall l :\ \mathcal{D}_{LA}(l,a) \\ \forall \langle w, b \rangle \in Z(\hat{s}) :\ \mathcal{D}_{WG}(w,g)}} x_{\hat{s}wblc} \leq \alpha_2^{ga} \qquad \begin{array}{c} \forall g \in \mathcal{D}_G \\ \forall a \in \mathcal{D}_A \end{array} \tag{17}$$

where $(\alpha_1^{ga}, \alpha_2^{ga})$ is the pair s.t. $\mathcal{D}_{GA\,\alpha_1\alpha_2}(g, a, \alpha_1^{lc}, \alpha_2^{lc})$ is true.

Constraints (8), (9) and (10) are implemented as:

$$\gamma_1 \leq \sum_{\forall \hat{s} \in \hat{S}\ \forall \langle w,b \rangle \in Z(\hat{s})\ \forall c \in \mathcal{D}_C} x_{\hat{s}wblc} \leq \gamma_2 \qquad \forall l \in \mathcal{D}_L \tag{18}$$

$$\tau_1^{br} \leq \sum_{\substack{\forall \hat{s} \in \hat{S}\ \forall c :\ \mathcal{D}_{CR}(c,r) \\ \forall w \in \mathcal{D}_W :\ \langle w, b \rangle \in Z(\hat{s})}} x_{\hat{s}wblc} \leq \tau_2^{br} \qquad \begin{array}{c} \forall b \in \mathcal{D}_B \\ \forall r \in \mathcal{D}_R \end{array} \tag{19}$$

$$\omega_1^{r} \leq \sum_{\forall \hat{s} \in \hat{S}\ \forall \langle w,b \rangle \in Z(\hat{s})\ \forall c :\ \mathcal{D}_{CR}(c,r)} x_{\hat{s}wblc} \leq \omega_2^{r} \qquad \forall r \in \mathcal{D}_R \tag{20}$$

where $(\tau_1^{br}, \tau_2^{br})$ and (ω_1^r, ω_2^r) are the pairs such that $\mathcal{D}_{BR\,\tau_1\tau_2}(b,\ r,\ \tau_1^{gr}, \tau_2^{gr})$ and $\mathcal{D}_{R\,\omega_1\omega_2}(r, \omega_1^r, \omega_2^r)$ are true.

Obviously equations (15)–(20) can be solved in time polynomial in the domain sizes. Nevertheless, there is a problem that must be dealt with: the equations could be too restrictive so that no solution exists. However, as we are interest in finding an approximate solution that has the minimal violation the problem can be solved by introducing two artificial variables for each equation. For instance, equation (18) is rewritten as:

$$w_1^l + \sum_{\forall \hat{s} \in \hat{S}\ \forall \langle w,b \rangle \in Z(\hat{s})\ \forall c \in \mathcal{D}_C} x_{\hat{s}wblc} \geq \gamma_1 \qquad \forall l \in \mathcal{D}_L$$

$$w_2^l - \sum_{\forall \hat{s} \in \hat{S}\ \forall \langle w,b \rangle \in Z(\hat{s})\ \forall c \in \mathcal{D}_C} x_{\hat{s}wblc} \geq -\gamma_2 \qquad \forall l \in \mathcal{D}_L$$

where w_1^l and w_2^l are non-negative artificial variables. We take the summation of all artificial variables as an objective function to be minimized. Therefore we obtain a linear program representation of the equation resolution problem. By solving the linear program we compute a solution minimizing the objective function – that is, the one with the minimal constraint violation.

We point out that the size of the linear program is not exponential as for the $\texttt{IFM}_\texttt{I}$ linear program but polynomial. Nevertheless, the number of both rows and columns may be large. So, a row-column generation approach can be adopted: at each iteration step, the pricing problem has to also decide which row must be eventually expanded.

Once solved the linear equation system, we have to perform the last task (6) to construct a feasible instance r. To this end, for each positive basic variable $x_{\hat{s}wblc}$, say with value k, we select k distinct values $s_1, \ldots, s_k$ from the domain of S, add the values of non-basic domains R, T, A and G, compute the measure P and insert the so constructed tuples into r.

4 Conclusions

Inverse OLAP is the problem of deciding whether a given fact table satisfies a set of given count constraints [12]. In this paper we have defined a special case of this problem, called Inverse Tree-OLAP, for which the flat fact table key is modeled by a Dimensional Fact Model (DFM) with a tree structure. The count constraints define aggregation patterns to be respected by both the many-to-many relationship among the basic dimensions and the one-to-many relationships within dimension hierarchies.

References

1. Arvind Arasu, Raghav Kaushik, and Jian Li. Data generation using declarative constraints. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 685–696, New York, NY, USA, 2011. ACM.

2. Catriel Beeri and Moshe Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37:15–46, 1990.
3. Toon Calders. The complexity of satisfying constraints on databases of transactions. *Acta Inf.*, 44(7-8):591–624, 2007.
4. George B. Dantzig and Mukund N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer-Verlag, 2006.
5. Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: A conceptual model for data warehouses. *Int. J. Cooperative Inf. Syst.*, 7(2-3):215–247, 1998.
6. Antonella Guzzo, Luigi Moccia, Domenico Saccà, and Edoardo Serra. Solving inverse frequent itemset mining with infrequency constraints via large-scale linear programs. *TKDD*, 7(4):18:1–18:39, 2013.
7. Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, 1996.
8. Svetlana Mansmann, Nafees Ur Rehman, Andreas Weiler, and Marc H. Scholl. Discovering olap dimensions in semi-structured data. In Il-Yeol Song and Matteo Golfarelli, editors, *DOLAP*, pages 9–16. ACM, 2012.
9. T. Mielikainen. On inverse frequent set mining. In IEEE Computer Society, editor, *Proc. of 2nd Workshop on Privacy Preserving Data Mining (PPDM)*, pages 18–23, 2003.
10. Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
11. Domenico Saccà, Edoardo Serra, and Alfredo Cuzzocrea, editors. *IDEAS 2018: 22nd International Database Engineering and Applications Symposium, June 1820, 2018, Villa San Giovanni, Italy*. ACM, 2018.
12. Domenico Saccà, Edoardo Serra, and Antonella Guzzo. Count constraints and the inverse olap problem: Definition, complexity and a step toward aggregate data exchange. In *FoIKS*, pages 352–369, 2012.
13. Gerhard Weikum. Where's the Data in the Big Data Wave? 2013. *ACM Sigmod BLOG:* http://wp.sigmod.org/?p=786.