Multi-Sorted Inverse Frequent Itemsets Mining: On-Going Research

Domenico Saccà *, Edoardo Serra, and Antonio Piccolo

[1] DIMES Department, University of Calabria, and ICT-SUD, Italy
sacca@unical.it
[2] Computer Science Department, Boise State University, USA
edoardoserra@boisestate.edu
[3] DIMES Department, University of Calabria, Italy
piccolo@dimes.unical.it

Abstract. Inverse frequent itemset mining (IFM) consists of generating artificial transactional databases reflecting patterns of real ones, in particular, satisfying given frequency constraints on the itemsets. An extension of IFM called *many-sorted* IFM, is introduced where the schemes for the datasets to be generated are those typical of Big Tables, as required in emerging big data applications, e.g., social network analytics.

Keywords: Big Data Synthesized Datasets, Inverse data mining, Frequent Itemsets

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 [11]. 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. As real-life workloads are often proprietary and out of reach for academic research, 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 even directly provided by domain experts) and then used to generate "realistic" privacy-preserving datasets.

In this paper we propose to extend inverse frequent itemset mining (IFM) that consists of generating a transactional database satisfying given support constraints on the itemsets in an input set, that are typically the frequent ones. We recall that frequent itemset mining is a popular mining task over transaction databases is to single out the set of the *frequent/infrequent itemsets* [1, 7, 10, 6] – some basic notation on IFM is presented in Section 2.

In order to enlarge the application domain of IFM, in Section 3 we introduce an extension of $\texttt{IFM}_\texttt{I}$, called ms-IFM, which considers more structured schemes for the datasets to be generated, as required in emerging big data applications, e.g., social network analytics. We present an overview of our on-going research work on how to solve ms-IFM and we draw the conclusion in Section 4.

* This work was carried out within the Cyber Security District, funded by MIUR.

2 Preliminaries and Related Work

Let $\mathcal{I}$ be a finite domain of n elements, also called *items*. Any subset $I \subseteq \mathcal{I}$ is an *itemset* over $\mathcal{I}$. A (*transactional*) *database* $\mathcal{D}$ over $\mathcal{I}$ (also called *dataset*) is a bag of itemsets, which may occur duplicated in $\mathcal{D}$ — the size $|\mathcal{D}|$ of $\mathcal{D}$ is the total number of its itemsets, called *transactions*.

Given a database $\mathcal{D}$ over $\mathcal{I}$, for each itemset $I \in \mathcal{D}$, there exist two important measures: (i) the *number of duplicates* of I, denoted as $\delta^{\mathcal{D}}(I)$, that is the number of occurrences of I in $\mathcal{D}$, and (ii) the *support* of I, denoted as $\sigma^{\mathcal{D}}(I)$, that is the sum of all number of duplicates of itemsets in $\mathcal{D}$ containing I, i.e., $\sigma^{\mathcal{D}}(I) = \sum_{J \in \mathcal{D} \wedge I \subseteq J} \delta^{\mathcal{D}}(J)$ – an alternative measure is the frequency $f^{\mathcal{D}}(I) = \sigma^{\mathcal{D}}(I)/|\mathcal{D}|$. A database $\mathcal{D}$ can be represented in a succinct format as a set of pairs $(I, \delta^{\mathcal{D}}(I))$.

We say that I is a *frequent* (resp., *infrequent*) itemset in $\mathcal{D}$ if its support is greater than or equal to (resp., less than) a given threshold.

The perspective of the frequent itemset mining problem can be naturally inverted: we are be given in advance a set of itemsets together with their frequency constraints and our goal is then to decide whether there is a transaction database satisfying the above constraints. This problem, called the *inverse frequent itemset mining* problem (IFM), has been introduced in the context of defining generators for benchmarks of mining algorithms [12], and has been subsequently reconsidered in privacy preserving contexts [2]). IFM has been proved to be in PSPACE and NP-hard. Observe that the original IFM formulation does not introduce any constraint on infrequency. A reformulation of IFM in terms of frequencies instead of supports has been introduced in [3, 4].

A drawback of IFM is that a solution may contain itemsets that are not expected to be frequent. A simple solution to exclude such itemsets from a feasible solution has been first proposed in [9] – the decision complexity of this problem is NP-complete. An elaborated version of IFM with infrequency support constraint (IFM_I for short), has been recently proposed in [8] and its decision complexity is NEXP-complete.

3 Many-Sorted IFM for Big Data Applications

Let a NoSQL *relation* $R(K, A_1, \ldots, A_p, A_{p+1}, \ldots, A_{p+q})$ be given, where K is the table key, $A_1, \ldots, A_p$ are classical single-valued (SV) attributes and $A_{p+1}, \ldots, A_{p+q}$ are multi-valued (MV) attributes. We assume that the attributes are ordered, i.e., K is the first attribute, A_1 the second attribute, A_{p+1} the $(1 + p + 1)$-th attribute and so on.

For each $i, 1 \leq i \leq p + q$, let $\mathcal{A}_i$ be the finite domain for the attributes A_i and $|\mathcal{A}_i| = n_i$. We assume that the values of every domain $\mathcal{A}_i$ (called *items*) are given in input – they are SV items or MV items depending on whether the attribute A_i is SV or MV. On the other hand, the domain $\mathcal{K}$ of the key K is countably infinite and, then, its values are not listed. Let $\dot{n} = \sum_{i=1}^{p} n_i$, $\ddot{n} = \sum_{i=p+1}^{p+q} n_i$ and $n = \dot{n} + \ddot{n}$.

A NoSQL *tuple* on R is of the form $t = [k, a_1, \ldots, a_p, g_1, \ldots, g_q]$, where $k \in \mathcal{K}$, for each i, $1 \leq i \leq p$, $a_i \in \mathcal{A}_i$ (a_i is an *item* of t) and for each i, $1 \leq i \leq q$, $g_i \subseteq \mathcal{A}_{p+i}$ (g_i is an *itemset* of t). Items and itemsets are called *values* of t. A NoSQL *table* on R is a finite set of NoSQL tuples.

A *many-sorted transaction* T is a $(p+q)$-tuple $[a_1, \ldots, a_p, g_1, \ldots, g_q]$ where the key attribute value is dropped out from a NoSQL tuple. It follows that a many-sorted transaction can be transformed into a tuple simply by inventing a key for it. Given any attribute A_i in R and a many-sorted transaction T, T_{A_i} denotes the value of T for the attribute A_i (e.g., $a_i \in \mathcal{A}_i$ if $i \leq p$ or $g_{i-p} \subseteq \mathcal{A}_i$ if $i > p$).

Let $\mathcal{T}$ be the set of all many-sorted transactions. The cardinality of $\mathcal{T}$ is $n_{\mathcal{T}} = 2^{\ddot{n}} \cdot \prod_{i=1}^{p} \dot{n}_i$. A *many-sorted dataset* $\mathcal{D}$ is set of pairs $(T, \delta^{\mathcal{D}}(T))$, where T is a many-sorted transaction and $\delta^{\mathcal{D}}(T)$ is the number of occurrences of T. $\mathcal{D}$ can be transformed into a NoSQL table $T_{\mathcal{D}}$ by making $\delta^{\mathcal{D}}(T)$ tuple duplicates of each many-sorted transaction T. The *cardinality* of $\mathcal{D}$, denoted as $|\mathcal{D}|$, is the number of pairs in $\mathcal{D}$ and the *size* of $\mathcal{D}$ is $\delta^{\mathcal{D}} = |T_{\mathcal{D}}| = \sum_{T \in \mathcal{D}} \delta^{\mathcal{D}}(T)$. We stress that in general $\delta^{\mathcal{D}} \gg |\mathcal{D}|$. From now on, we shall omit the term *many-sorted* whenever it is clear from the context.

A *sub-transaction* S is a $(p+q)$-tuple $[a_1, \ldots, a_p, g_1, \ldots, g_q]$ on R for which the domain of each SV attribute is extended with $\perp$, which stands for a null value. Let $\perp(S)$ denote the number of null values in S – a transaction T can be also seen as a sub-transaction type for which $\perp(T) = 0$. The *length* of S, denoted as $l(S)$, is the number of values different from $\perp$ and $\emptyset$.

A sub-transaction S *subsumes* a transaction T (written $S \sqsubseteq T$) if for each SV attribute A_i, either $S.A_i = \perp$ or $S.A_i = T.A_i$, and for each MV attribute A_i, $S.A_i \subseteq T.A_i$. Observe that if S happens to be a transaction then every transaction T for which $S \sqsubseteq T$ has the same SV items as S, whereas its MV itemsets are supersets of the corresponding itemsets in S. In the classical `IFM` setting, every transaction type S coincides with an itemset and the transactions T subsumed by S are all itemsets for which $S \subseteq T$. This analogy explains why we write $S \sqsubseteq T$ for transaction subsumption.

Given a sub-transaction S for which $l(S) > 0$ and two integers σ_1 and σ_2 for which $0 \leq \sigma_1 \leq \sigma_2$, $\gamma = \langle S, \sigma_1, \sigma_2 \rangle$ represents a *frequency support constraint* defined as: a database $\mathcal{D}$ satisfies γ (written as $\mathcal{D} \models \gamma$) if: $\sigma_1 \leq \sum_{T \in \mathcal{D} \wedge S \sqsubseteq T} \delta^{\mathcal{D}}(T) \leq \sigma_2$.

An *infrequency support constraint* γ is a pair $\langle S, \sigma \rangle$ and is actually a shorthand for the frequency support constraint $\langle S, 0, \sigma \rangle$. The upper bound σ will be simply referred to as $\gamma_{\#}$. A (frequency or infrequency) support constraint γ for which $l(\gamma_S) = 1$ is called a *domain support constraint*.

Given a set Π of support constraints, a database $\mathcal{D}$ satisfies Π (written as $\mathcal{D} \models \Pi$), if for each $\gamma \in \Pi$, $\mathcal{D} \models \gamma$.

Example 1. Individuals are characterized by the SV attributes *Gender*, *Location* and *Age* and by the MV attributes *Groups* and *Events*: an individual may belong to various groups and may attend a number of events. A transaction $I = [Male, Rome, 25, \{g_1, g_4\}, \{e_1, e_3\}]$ represents a 25-year old male individual located in Rome who belongs to the groups g_1 and g_4 and attends the events e_1 and e_3. The transaction $J = [Female, Rome, 20, \{g_1, g_2\}]$ represents an individual who does not attend any event. Note that, as the attributes do not define a key, there may exist several occurrences of the same individual. Examples of constraints are shown next.

Frequency constraints:

– $\langle [Male, Rome, \perp, \{g_1, g_2\}, \emptyset], 10000, 20000 \rangle$ fixes the range for the overall duplicate number of male individuals who are located in Rome and are participating to at least the groups g_1 and g_2;

– $\langle[Female, \bot, 25, \{g_1,\ g_2\}, \{e_1, e_3\}], 500, 1000\rangle$ fixes the range for the overall duplicate number of 25-year old female individuals who are participating to at least the groups g_1 and g_2 and attending at least the events e_1 and e_3;
Infrequency constraints:
– $\langle[\bot, Cosenza, \bot, \{g_1, g_2\}, \emptyset], 100\rangle$ states that the number of individuals located at Cosenza in a feasible dataset who are participating to at least the groups g_1 and g_2 is at most 100;
– $\langle[\bot, \bot, \bot, \emptyset, \{e_1, e_2\}], 10000\rangle$ states that the number of individuals attending at least the events e_1 and e_2 is at most 10000;
– $\langle[\bot, \bot, \bot, \{g_1\}, \emptyset], 100000\rangle$ is a domain support constraint stating that the number of individuals participating to at least the group g_1 is at most 100000. □

From now on, we assume that the following sets of constraints are given: (1) set Σ of *frequency support constraints* with cardinality $m = |\Sigma| > 0$ and (2) a set $\widehat{\Sigma}$ of *infrequency support constraints* with cardinality $\widehat{m} = |\widehat{\Sigma}| \geq 0$. Let $\breve{\Sigma} = \Sigma \cup \widehat{\Sigma}$ and $\breve{m} = |\breve{\Sigma}| = m + \widehat{m}$.

Given R, $\breve{\Sigma} = \Sigma \cup \widehat{\Sigma}$, and an integer $\texttt{size} > 0$, the *multi-sorted inverse frequent itemset mining problem*, shortly denoted as ms–IFM, consists of finding a many-sorted dataset $\mathcal{D}$ on R such that both $\delta^{\mathcal{D}} = \texttt{size}$ and $\mathcal{D} \models \breve{\Sigma}$ (or of eventually stating that there is no such a dataset).

By assuming that items, constraint bounds and size are stored using a constant amount of space, the input size of the problem is $\mathcal{O}(\, n + \breve{m}\,(p + q\,\ddot{n} + 1) + m\,)$.

It is easy to see that ms–IFM reduces to the classical IFM problem if $p = 0$ and $q = 1$, i.e., there exists exactly one non-key attribute in R and this attribute is of MV type, say G. A transaction is then any itemset on the domain $\mathcal{G}$ of G. The next result shows that the complexity of ms–IFM.

The decision version of ms–IFM is in PSPACE and NP-hard. To provide a more efficient resolution algorithm for the problem, we relax the integer constraint for the number of duplicates for a transaction of a database $\mathcal{D}$, i.e., it may be a rational number. We call *relaxed* ms–IFM this version of the problem. Then the decision version of relaxed ms–IFM is NP-complete.

Note that, as shown in [8], the higher complexity of the $\texttt{IFM}_\texttt{I}$ problem derives from the task of discovering "minimal" itemsets that must be enforced to be infrequent. Instead such infrequent itemsets are assumed to be part of the input for ms–IFM.

4 Notes on ms-IFM Resolution and Conclusions

Following the approach of [8], we formulate the ms–IFM problem as a linear program (LP) with a linear number of constraints (corresponding to the support constraints of the itemsets) and an exponential number variables, one for each possible transaction, indicating the number of its occurrences in the target database. We represent the LP in a succinct format with size $\mathcal{O}(n\,m)$, where n is the total number of items and m is the total number of constraints.

In [8] the simplex method approach is adopted to solve our LP problem for it continues to be effectively and efficiently used to solve linear programs notwithstanding it may get exponential time. The main issue dealt with is to apply the simplex to a LP with

an exponential number of variables. The literature gives an interesting solution: *column generation*, see e.g [5], that 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*. As the decision version of the latter problem is proven to be NP complete for $\texttt{IFM}_\texttt{I}$, a heuristic algorithm for the pricing problem is used. An exact exponential time pricing algorithm is presented as well, which is only executed at the last iterations of column generation A large number of experiments have confirmed that capability of the approach to solve large instances of both synthesized and real datasets.

Our on-going research is focused in adapting the approach of [8] to solve $ms\text{–}\texttt{IFM}$ by providing suitable new solvers for the pricing problem: an exact solver and a heuristic one.

References

1. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 439–450, New York, NY, USA, 2000. ACM.
3. T. Calders. Computational complexity of itemset frequency satisfiability. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 143–154, New York, NY, USA, 2004. ACM.
4. T. Calders. The complexity of satisfying constraints on databases of transactions. *Acta Informatica*, 44(7–8):591–624, 2007.
5. G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer-Verlag, 2006.
6. B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on fimi'03. *SIGKDD Explorations Newsletter*, 6(1):109–117, 2004.
7. D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In A. O. Mendelzon and Z. M. Özsoyoglu, editors, *PODS'97*, pages 209–216. ACM Press, 1997.
8. A. Guzzo, L. Moccia, D. Saccà, and E. Serra. Solving inverse frequent itemset mining with infrequency constraints via large-scale linear programs. *TKDD*, 7(4):18, 2013.
9. A. Guzzo, D. Saccà, and E. Serra. An effective approach to inverse frequent set mining. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 806–811, Washington, DC, USA, 2009. IEEE Computer Society.
10. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
11. K. Michael and K. W. Miller. Big data: New opportunities and new challenges [guest editors' introduction]. *Computer*, 46(6):22–24, 2013.
12. T. Mielikainen. On inverse frequent set mining. In *Proceedings of 2nd Workshop on Privacy Preserving Data Mining*, PPDM '03, pages 18–23, Washington, DC, USA, 2003. IEEE Computer Society.
13. G. Weikum. Wheres the Data in the Big Data Wave? 2013. *ACM Sigmod BLOG:* http://wp.sigmod.org/?p=786.