Proceedings of the
1st International Conference and Exhibition on Future RFID Technologies
Eszterhazy Karoly University of Applied Sciences and
Bay Zoltán Nonprofit Ltd. for Applied Research
Eger, Hungary, November 5–7, 2014. pp. 161–169

doi: 10.17048/FutureRFID.1.2014.161

CGT: a vertical miner for frequent equivalence classes of itemsets

Laszlo Szathmary, Márton Ispány

University of Debrecen, Faculty of Informatics, Department of IT
{szathmary.laszlo,ispany.marton}@inf.unideb.hu

Abstract

In this paper we present a vertical, depth-first algorithm that outputs frequent generators (FGs) and their associated frequent closed itemsets (FCIs). The proposed algorithm –called *CGT*– is a single-pass algorithm and it explores frequent equivalence classes in a dataset.

1. Introduction

In data mining, frequent itemsets (FIs) and association rules play an important role [1]. Due to the high number of patterns, various concise representations of FIs have been proposed, of which the most well known representations are the FGs and the FCIs [2, 3]. There are a number of methods in the literature that target both FCIs and FGs, but most of these algorithms are levelwise methods [4, 5]. It is known that depth-first algorithms usually outperform their levelwise competitors. Here we present a single-pass, depth-first, vertical FG+FCI miner.

The remainder of the paper is organized as follows. Background on pattern mining and concept analysis is provided in Section 2. Section 3 presents our proposed algorithm *CGT* in detail, including pseudo code and running example. Conclusions and future work directions are given in Section 4.

2. Basic concepts

In the following, we recall basic concepts from frequent pattern mining and formal concept analysis (FCA). The vocabulary and notations come from the dedicated

literature but, whenever necessary, parallels are drawn to support the comprehension. The following 4×6 sample dataset: $\mathcal{D} = \{(1,\ ACDE),\ (2,\ ABCDE),\ (3,\ ABE),\ (4,\ BEF)\}$ will be used as a running example. Henceforth, we refer to it as dataset $\boldsymbol{\mathcal{D}}$.

We consider a set of *objects* or *transactions* $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* or *items* $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, and a relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$. A set of items is called an *itemset*. Each transaction has a unique identifier (*tid*), and a set of transactions is called a *tidset*. The tidset of all transactions sharing a given itemset X is its *image*, denoted $t(X)$. For instance, the image of $\{A, B\}$ in $\mathcal{D}$ is $\{2, 3\}$, i.e., $t(AB) = 23$ in our separator-free set notation. The *length* of an itemset is its cardinality, whereas an itemset of length k is called a k-itemset. The (absolute) *support* of an itemset X, denoted by $supp(X)$, is the size of its image, i.e. $supp(X) = |t(X)|$. An itemset X is called *frequent*, if its support is not less than a given *minimum support* (denoted by min_supp), i.e. $supp(X) \geq min\_supp$. An equivalence relation is induced by t on the power-set of items $\wp(\mathcal{A})$: equivalent itemsets share the same image ($X \cong Z$ iff $t(X) = t(Z)$). Consider the equivalence class of X, denoted $[X]$, and its extremal elements w.r.t. set inclusion. $[X]$ knowingly admits a unique maximum (a *closed* itemset), and a set of minimal elements (*generator* itemsets). The following definition thereof exploits the monotony of *supp* upon $\subseteq$ within $\wp(\mathcal{A})$:

Definition 2.1. An itemset X is *closed* (*a generator*) if it has no proper superset (subset) with the same support.

A *closure* operator underlies the set of closed itemsets; it assigns to X the maximum of $[X]$ (denoted by $\gamma(X)$). Naturally, $X = \gamma(X)$ for closed X. Generators, a.k.a. *key-sets* in database theory, represent a special case of free-sets [6]. For instance, in our dataset $\mathcal{D}$, B and C are generators, with closures BE and $ACDE$, respectively (see Figure 1).

In [7], a subsumption relation is defined as well: X *subsumes* Z, iff $X \supset Z$ and $supp(X) = supp(Z)$. By Def. 2.1, if Z *subsumes* X, then Z cannot be a generator.

The following property, which is part of the folklore in the domain, generalizes this observation. It basically states that the generator family forms a downset within the Boolean lattice $\langle \wp(\mathcal{A}), \subseteq \rangle$:

Property 2.2. *Given $X \subseteq \mathcal{A}$, if X is a generator, then $\forall Y \subseteq X$, Y is a generator. Equivalently, if X is not a generator, $\forall Z \supseteq X$, Z is not a generator.*

The FCI and FG families are well-known reduced representations [8] for FIs, which jointly compose non-redundant bases of valid association rules, e.g. the *generic basis* [3].

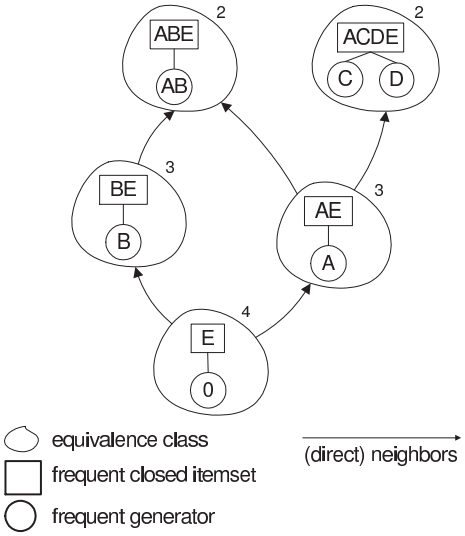

Figure 1: Equivalence classes of $\mathcal{D}$ with $min\_supp = 2$. Support values are shown in the top right-hand corner of the classes. The generator of E is the empty set.

3. Filtering generators and closed itemsets among frequent itemsets using a depth-first traversal

Our own algorithm *CGT*, which is the contribution of this paper, is a vertical itemset mining algorithm for finding frequent equivalence classes. In this section first we provide a general view of *CGT*. Then we give a background on vertical algorithms such as *Eclat* [9] and *Talky* [10]. Finally we detail *CGT*.

3.1. A general view of CGT

CGT is based on *Talky* [10], where *Talky* is a modified version of *Eclat* [9]. *Eclat* and *Talky* produce the same output, i.e. they find all FIs in a dataset. However, *Talky* uses a different traversal called reverse pre-order strategy. This traversal goes from right-to-left and it provides a special feature: when we reach an itemset X, all subsets of X were already discovered. As a result, this traversal can be used to filter FGs among FIs. During the traversal procedure *CGT* also filters FCIs and assigns them to the corresponding FGs, thus *CGT* outputs at the end the frequent equivalence classes. In order to filter the frequent generators, it must rely on the reverse pre-order strategy that we describe in the next subsections.

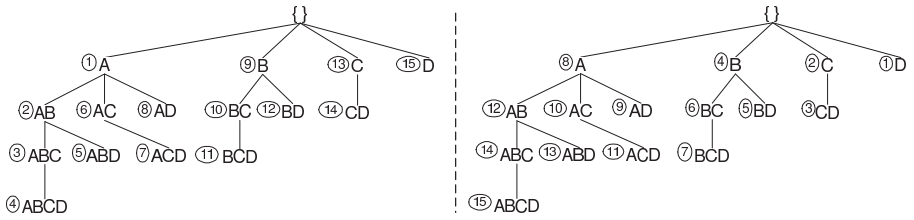

Figure 2: **Left:** pre-order traversal with *Eclat*; **Right:** reverse pre-order traversal with *Talky*. The direction of traversal is indicated in circles.

3.2. Vertical itemset mining

Miners from the literature, whether for plain FIs or FCIs, can be roughly split into breadth-first and depth-first ones. Breadth-first algorithms, more specifically the *Apriori*-like [1] ones, apply levelwise traversal of the pattern space exploiting the anti-monotony of the frequent status. Depth-first algorithms, e.g., *Closet* [11], in contrast, organize the search space into a prefix-tree (see Figure 2) thus factoring out the effort to process common prefixes of itemsets. Among them, the *vertical* miners use an encoding of the dataset as a set of pairs (item, tidset), i.e., $\{(i, t(i))|i \in \mathcal{A}\}$, which reportedly allows the costly database re-scans to be avoided.

Eclat [9] was the first FI-miner to combine the vertical encoding with a depth-first traversal of a tree structure, called IT-tree, whose nodes are $X \times t(X)$ pairs. *Eclat* traverses the IT-tree in a depth-first manner in a pre-order way, from left-to-right [9, 12] (see Figure 2).

3.3. Reverse pre-order traversal

CGT extends *Talky* to filter FGs and FCIs among FIs. That is, *CGT* tests every newly found FI if it is a generator. For the test to be effective, all subsets of a candidate X must be processed *before* X itself. Only then all generator subsets of X will be available for a thorough test of X being generator itself. Although such a concern is typically addressed through a breadth-first traversal strategy in mining, the same order could also be achieved with a depth-first one, yet with a different order on the items.

Traversing the search space so that a given set X is processed after all its subsets is a frequent requirement in combinatorial algorithms. Levelwise methods straightforwardly satisfy this condition. Following an idea in [13], called *reverse pre-order traversal*, we rank items in the initial ordering in reverse lexicographic order (E, D, C, etc.). Thus, following the increasing order of numerical equivalents, we get a depth-first right-to-left traversal of a prefix-tree representing the search space $\wp(\mathcal{A})$. As at all nodes corresponding sets are listed before the sets corresponding to descendant nodes, the processing is "pre" (rather than "post").

In summary, our method traverses the IT-tree in a pre-order way from right-

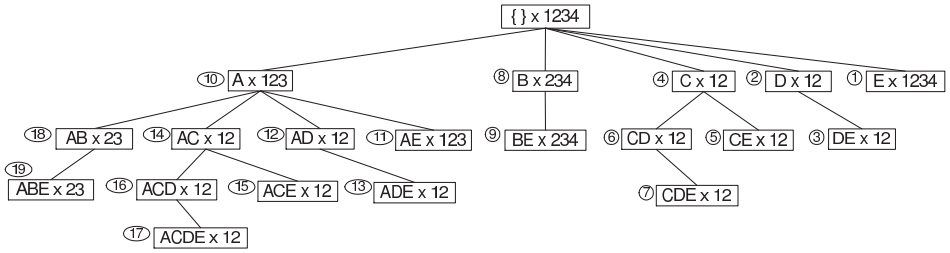

Figure 3: Execution of *Talky* on dataset $\mathcal{D}$ with $min\_supp = 2$. The processing order of nodes is indicated in circles.

Table 1: *CGT* builds this table, which is actually a hash table. Key of the hash: a tidset. Value of the hash: a row of the table.

tidset	generators	eq. class members (optional)	closure	support
1234	$\emptyset$	E	E	4
234	B	BE	BE	3
123	A	AE	AE	3
23	AB	ABE	ABE	2
12	D, C	$DE, CE, CD, CDE, AD, ADE, AC, ACE, ACD, ACDE$	$ACDE$	2

to-left. Thus, given an itemset X in a node in the IT-tree, it is guaranteed that the nodes corresponding to the subsets of X will be explored *before* X.

Example. See Figure 2 for a comparison between the two traversals namely pre-order with *Eclat* (left) and reverse pre-order with *Talky* (right).

3.4. Talky

Talky is a vertical FI miner that constructs an IT-tree in a depth-first manner in a reverse pre-order way (see Figure 3). From our dataset $\mathcal{D}$ with $min\_supp = 2$, *Talky* extracts the following 19 FIs in this order[1]: E (4), D (2), DE (2), C (2), CE (2), CD (2), CDE (2), B (3), BE (3), A (3), AE (3), AD (2), ADE (2), AC (2), ACE (2), ACD (2), $ACDE$ (2), AB (2) and ABE (2).

3.5. CGT in detail

In this subsection we present the *CGT* algorithm in detail. As mentioned before, *CGT* is based on *Talky*. *CGT* traverses the IT-tree in a reversed pre-order way (see Figure 3), and it filters FGs and FCIs while extracting FIs from a dataset. *CGT* groups generators to their closure, thus the output of *CGT* is the list of frequent equivalence classes (see Table 1).

[1]Support values are indicated in parentheses.

CGT builds a hash table[2], as depicted in Table 1. The key of the hash is a tidset, while the value of the hash is a row object. A row object represents an equivalence class and it has the following fields: **(1)** tidset (by definition all itemsets in an equivalence class have the same tidsets), **(2)** generators (minimal elements of an equivalence class), **(3)** equivalence class members (itemsets in an equivalence class that are neither generators nor closed itemsets), **(4)** closure (the largest element in an equivalence class; this is a unique element), **(5)** support (this is the cardinality of the tidset).

The algorithm works the following way. When a new FI is found in the IT-tree, it is tested if it belongs to an already discovered equivalence class, i.e. we test if its tidset is in the hash. If it is not present in the hash, then it belongs to a new equivalence class, thus a new row is added to the hash. If its tidset is in the hash, then there are two possible cases. Let R denote the row whose tidset is the same as the tidset of the current itemset, i.e. R represents the equivalence class where the current itemset belongs to.

Case 1. The itemset has a proper subset in the "generators" field of row R. It means that the itemset is not a generator, but it belongs to this equivalence class. The itemset is added to the "closure" field. The "closure" is the union of the generators and the itemsets that belong to the same equivalence class. Since *CGT* finds all FIs, it discovers all the members of an equivalence class, thus if we take the unions of all the members of an equivalence class, the closure of the equivalence class will be found correctly. Optionally, non-generator members of an equivalence class can be stored in the "eq. class members" field. This field is indicated in Table 1 for an easier understanding, but in the implementation it can be omitted. A non-generator member of an equivalence class can be added directly to the "closure" field using the union operation, it does not need to be stored separately.

Case 2. The itemset does not have a proper subset in the "generators" field of row R. It means the itemset is a new generator of the equivalence class, thus it is added to the "generators" field. Whenever a new generator is registered, it is also added to the "closure" field, where "closure" is the union of the added itemsets.

When the algorithm stops, the itemsets in the "closure" field are completed, i.e. they represent the closures of the equivalence classes. The pseudo code of *CGT* is provided in Algorithm 1.

3.5.1. Running example

Our dataset $\mathcal{D}$ is somewhat special since its column E is full. It means that E is not a generator because it has a proper subset with the same support namely the empty set. By definition, the support of the empty set is 100%. Thus, the hash table is initialized as seen in Table 2. Actually, the hash can be initialized this way with any dataset, but if the dataset has no full column, then the closure of the empty set will remain the empty set.

[2]In our implementation we used the `java.util.HashMap` class.

Algorithm 1 (pseudo code of CGT):

hashTable: the table structure (as seen in Table 1)

```
1)  // initialization
2)  row.tidset ← {largest tidset}   // in our example: 1234
3)  row.generators.add(∅)   // add the empty set
4)  row.support ← cardinality(row.tidset)
5)  hashTable.add(row)
6)
7)  // main block
8)  start the Talky algorithm and assign the current node to the variable curr
9)  {
10)     if curr.tidset not in hashTable:
11)         row.tidset ← curr.tidset
12)         row.generators.add(curr.itemset)
13)         row.closure ← curr.itemset
14)         row.support ← cardinality(row.tidset)
15)         hashTable.add(row)
16)     else:
17)         row ← hashTable.get(curr.tidset)
18)         if curr.itemset has a proper subset in row.generators:
19)             row.eq_class_members.add(curr.itemset)   // optional
20)             row.closure ← row.closure ∪ curr.itemset
21)         else:
22)             row.generators.add(curr.itemset)
23)             row.closure ← row.closure ∪ curr.itemset
24) }
25) // hashTable is filled; it contains all the frequent equivalence classes
```

Then, the algorithm starts enumerating the 19 FIs of $\mathcal{D}$ using the traversal strategy of *Talky* (see Section 3.4 for the list of FIs in $\mathcal{D}$). The first node is $E \times 1234$. The tidset 1234 is an existing key in the hash. E has a proper subset with the same support (the empty set), thus E is added to the "eq. class members" and "closure" fields. The next FI is $D \times 12$. Since 12 is not yet in the hash, a new row is added in the hash table. The next node is $DE \times 12$. The tidset 12 is in the hash, thus DE belongs to an existing equivalence class. It has a proper subset, D, thus DE is added to the "eq. class members" and "closure" fields. The current state of the hash table is depicted in Table 3.

We skip the step by step presentation of the rest of the algorithm. The end result of CGT is shown in Table 1.

Table 2: Initialization of CGT's hash table

tidset	generators	eq. class members (optional)	closure	support
1234	$\emptyset$		$\emptyset$	4

Table 3: State of CGT's hash table after adding E, D and DE

tidset	generators	eq. class members (optional)	closure	support
1234	$\emptyset$	E	E	4
12	D	DE	DE	2

4. Conclusion and future work

In this paper we presented a vertical, depth-first algorithm that outputs FG/FCI pairs and thus basically pinpoints the borders of frequent equivalence classes. The CGT algorithm was thought as a first step towards the design of a single-pass vertical FG+FCI miner, hence it represents an adaptation of a plain FI miner from the literature: efficient filtering for FCI and FGs are added whereby the FG-to-FCI association is immediate. Thus, CGT can be easily upgraded to a complete solution for the association rule base construction, e.g., by combining it with a precedence computing algorithm such as *Snow* [14].

In the near future, we shall concentrate on various strategies for reducing the traversal effort in CGT as well as speeding-up the computing of FCIs from members of their respective equivalence classes. One track would be to use counting inference while another one leads to the definition of a canonical FG in each class, to focus on for closure computation.

Acknowledgements. The publication work was supported by the TAMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

References

[1] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proc. of the 20th Intl. Conf. on Very Large Data Bases (VLDB '94), San Francisco, CA, Morgan Kaufmann (1994) 487–499

[2] Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. In: Proc. of the Computational Logic (CL '00). Volume 1861 of LNAI., Springer (2000) 972–986

[3] Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery. (2002) 92–109

[4] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proc. of the 7th Intl. Conf. on Database Theory (ICDT '99), Jerusalem, Israel (1999) 398–416

[5] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing Iceberg Concept Lattices with Titanic. Data and Knowl. Eng. **42**(2) (2002) 189–222

[6] Boulicaut, J.F., Bykowski, A., Rigotti, C.: Free-Sets: A Condensed Representation of Boolean Data for the Approximation of Frequency Queries. Data Mining and Knowledge Discovery **7**(1) (Jan 2003) 5–22

[7] Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: SIAM Intl. Conf. on Data Mining (SDM' 02). (Apr 2002) 33–43

[8] Calders, T., Rigotti, C., Boulicaut, J.F.: A Survey on Condensed Representations for Frequent Sets. In Boulicaut, J.F., Raedt, L.D., Mannila, H., eds.: Constraint-Based Mining and Inductive Databases. Volume 3848 of Lecture Notes in Computer Science., Springer (2004) 64–80

[9] Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proc. of the 3rd Intl. Conf. on Knowledge Discovery in Databases. (August 1997) 283–286

[10] Szathmary, L., Valtchev, P., Napoli, A., Godin, R.: Efficient Vertical Mining of Frequent Closures and Generators. In: Proc. of the 8th Intl. Symposium on Intelligent Data Analysis (IDA '09). Volume 5772 of LNCS., Lyon, France, Springer (2009) 393–404

[11] Pei, J., Han, J., Mao, R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. (2000) 21–30

[12] Zaki, M.J.: Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering **12**(3) (2000) 372–390

[13] Calders, T., Goethals, B.: Depth-first non-derivable itemset mining. In: Proc. of the SIAM Intl. Conf. on Data Mining (SDM '05), Newport Beach, USA. (Apr 2005)

[14] Szathmary, L., Valtchev, P., Napoli, A., Godin, R., Boc, A., Makarenkov, V.: A Fast Compound Algorithm for Mining Generators, Closed Itemsets, and Computing Links Between Equivalence Classes. Annals of Mathematics and Artificial Intelligence (AMAI) (Aug 2013) 1–25