Annales Mathematicae et Informaticae
***48** (2018) pp. 75–82*
http://ami.uni-eszterhazy.hu

Finding frequent closed itemsets with an extended version of the Eclat algorithm

Laszlo Szathmary

University of Debrecen, Faculty of Informatics, Department of IT
H-4002 Debrecen, Pf. 400, Hungary
szathmary.laszlo@inf.unideb.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Apriori is the most well-known algorithm for finding frequent itemsets (FIs) in a dataset. For generating interesting association rules, we also need the so-called frequent closed itemsets (FCIs) that form a subset of FIs. *Apriori* has a simple extension called *Apriori-Close* that can filter FCIs among FIs. However, it is known that vertical itemset mining algorithms outperform the *Apriori*-like levelwise algorithms. *Eclat* is another well-known vertical miner that can produce the same output as *Apriori*, i.e. it also finds the FIs in a dataset. Here we propose an extension of *Eclat*, called *Eclat-Close* that can filter FCIs among FIs. This way *Eclat-Close* can be used as an alternative of *Apriori-Close*. Experimental results show that *Eclat-Close* performs much better than *Apriori-Close*, especially on dense, highly-correlated datasets.

Keywords: data mining, frequent itemsets, association rules, algorithms

MSC: 97R40

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 frequent generators (FGs) and the frequent closed itemsets (FCIs) [2, 3]. There are a number of methods in the literature that target FCIs and/or FGs, but most of these

algorithms are levelwise methods [4, 5]. It is known that depth-first algorithms usually outperform their levelwise competitors.

In this paper, we present an algorithm called *Eclat-Close*, which is a single-pass, depth-first, vertical FI+FCI miner. The approach behind *Eclat-Close* is derived from the one used in the *Eclat* [6] vertical miner. *Eclat* outputs the entire family of FIs hence what we needed to design was a mechanism to recognize FCIs among FIs. The task performed by *Eclat-Close* can be described as the computation of frequent equivalence classes. The nice surprise with *Eclat-Close* came when we measured its performance. Despite its relatively low level of optimization, our algorithm systematically outperformed its levelwise competitor, *Apriori-Close*.

The remainder of the paper is organized as follows. Basic concepts are provided in Section 2. Section 3 presents the *Apriori-Close* algorithm, which is our levelwise competitor. Section 4 introduces *Eclat-Close*, which is the main contribution of the paper. Experimental results are provided in Section 5. Finally, Section 6 concludes the paper.

2. Basic Concepts

The following 5×5 sample dataset: $\mathcal{D} = \{(1,\ ABDE),\ (2,\ AC),\ (3,\ ABCE),\ (4,\ BCE),\ (5,\ ABCE)\}$ 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 by $t(X)$. For instance, the image of $\{A, B\}$ in $\mathcal{D}$ is $\{1, 3, 5\}$, i.e., $t(AB) = 135$ in our separator-free set notation. The *length* of an itemset X is $|X|$, whereas an itemset of length i is called an i-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 $[X]$. The equivalence class $[X]$ has a unique maximum w.r.t. set inclusion (a *closed* itemset).

Definition 2.1. An itemset X is *closed* if it has no proper superset 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. For instance, in our dataset $\mathcal{D}$, the closure of B is BE, while the closure of BC is BCE.

3. Levelwise Exploration with Apriori-Close

The most well-known levelwise algorithm, without doubt, is *Apriori* [7]. This algorithm addresses the problem of finding all frequent itemsets in a dataset. *Apriori* has been followed by lots of variations, and several of these levelwise algorithms concentrate on a special subset of frequent itemsets, like closed itemsets or generators. Mannila and Toivonen provided a general framework for levelwise algorithms in [8]. The levelwise algorithm for finding all FIs is a breadth-first, bottom-up algorithm, which means the following. First it finds all 1-long frequent itemsets[1], then at each i^{th} iteration it identifies all i-long frequent itemsets. The algorithm stops when it has identified the largest frequent itemset. Frequent itemsets are computed iteratively, in ascending order by their length. At each iteration one database pass is needed to count support values, thus the number of database passes is equal to the length of the largest frequent itemset. This approach is very simple and efficient for sparse, weakly correlated data. The levelwise algorithm is based on two basic properties.

Property 3.1 (downward closure)**.** *All subsets of a frequent itemset are frequent.*[2]

Property 3.2 (anti-monotonocity)**.** *All supersets of a non-frequent itemset are non-frequent.*

Apriori-Close was proposed in [9]. This algorithm is an extension of *Apriori* and it can identify not only frequent, but *frequent closed itemsets* too simultaneously. The idea is the following. By definition, a closed itemset has no proper superset with the same support. At each i^{th} step all i-long frequent itemsets are marked as "closed". At the next $(i+1)^{th}$ iteration for each $(i+1)$-long itemset we test if it has an i-long subset with the same support. If so, then the i-long frequent itemset is not a closed itemset and we mark it as "not closed". When the algorithm terminates with the enumeration of all frequent itemsets, the itemsets still marked as "closed" are the frequent closed itemsets of the dataset. Experiments show that this kind of filtering of closed itemsets does not induce any serious additional computation time.

4. The Eclat-Close Algorithm

In this section we present the *Eclat* algorithm [6], which serves as a basis for *Eclat-Close*. *Eclat* can only find FIs, while *Eclat-Close* makes it possible to filter FCIs among FIs. *Eclat-Close* is our extension and this section is the main contribution of the paper.

[1] That is, first it identifies all frequent items (attributes).

[2] The name of the property comes from the fact that the set of frequent itemsets is closed w.r.t. set inclusion.

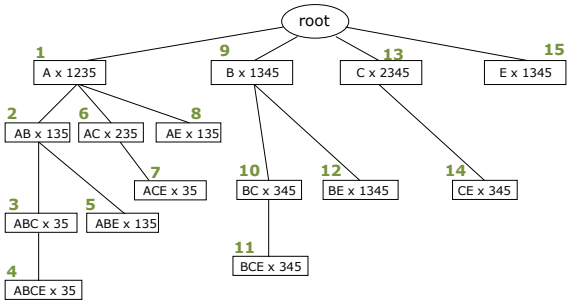

Figure 1: IT-tree: Itemset-Tidset search tree of dataset $\mathcal{D}$ with $min\_supp = 2$

4.1. Eclat

Eclat was the first FI-miner using a vertical encoding of the database combined with a depth-first traversal of the search space (organized in a prefix-tree) [6].

Vertical miners rely on a specific layout of the database that presents it in an item-based, instead of a transaction-based, fashion. Thus, an additional effort is required to transpose the global data matrix in a pre-processing step. However, this effort pays back since afterwards the secondary storage does not need to be accessed anymore. Indeed, the support of an itemset can be computed by explicitly constructing its tidset which in turn can be built on top of the tidsets of the individual items. Moreover, in [10], it is shown that the support of any k-itemset can be determined by intersecting the tid-lists of any two of its $(k-1)$-long subsets.

The central data structure in a vertical FI-miner is the IT-tree that represents both the search space and the final result. The IT-tree is an extended prefix-tree whose nodes are $X \times t(X)$ pairs. With respect to a classical prefix-tree or trie, in an IT-tree the itemset X provides the entire prefix from the root to the node labeled by it (and not the difference with the parent node prefix).

Example. Figure 1 presents the IT-tree of our example. The traversal order is indicated above the nodes. Observe that the node $ABC \times 35$ for instance can be computed by combining the nodes $AB \times 135$ and $AC \times 235$. To that end, tidsets are intersected and itemsets are joined. The support of ABC is readily established to 2.

4.2. Eclat-Close

In this subsection we present the *Eclat-Close* algorithm in detail. As mentioned before, *Eclat-Close* is based on *Eclat*. *Eclat-Close* traverses the IT-tree in a pre-order way, from left to right (see Figure 1), and it filters FCIs while extracting FIs from a dataset. The output of *Eclat-Close* is the list of frequent equivalence classes (see Table 1).

tidset	eq. class members (optional)	closure	support
1235	A	A	4
135	AB, ABE, AE	ABE	3
35	ABC, $ABCE$, ACE	$ABCE$	2
235	AC	AC	3
1345	B, BE, E	BE	4
345	BC, BCE, CE	BCE	3
2345	C	C	4

Table 1: *Eclat-Close* builds this table, which is actually a hash table. The key is a tidset and the value is a row

Eclat-Close builds a hash table, 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 tidset), **(2)** equivalence class members, **(3)** closure (the largest element in an equivalence class; this is a unique element), and **(4)** 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 the following steps are performed. First, the itemset is added to the row's list of equivalence class members. Second, the itemset is added to the row's closure using a union operation.

Example. *Eclat-Close* builds a hash table, as depicted in Table 1. A row object represents an equivalence class. The algorithm starts enumerating the 15 FIs of $\mathcal{D}$ using the traversal strategy of *Eclat* (as seen in Figure 1). The first node is $A \times 1235$. The tidset 1235 is not yet in the hash, thus a new row is added in the hash table (tidset: 1235; eq. class members: A; closure: A; support 4). The nodes $AB \times 135$ and $ABC \times 35$ are also added as new rows. The next FI is $ABCE \times 35$, but its tidset is an existing key in the hash. Let r denote the row whose tidset is 35. $ABCE$ is added to r's "eq. class members" and "closure" fields. The "closure" column is the union of its former value ABC and $ABCE$, which yields $ABCE$. The end result is shown in Table 1.

When the algorithm stops, the itemsets in the "closure" field are completed, i.e. they represent the closures of the equivalence classes. If we are only interested in FCIs, the column "eq. class members" can be omitted. This way *Eclat-Close* can be used as a pure FCI-miner algorithm. The pseudo code of *Eclat-Close* is provided in Algorithm 1.

Algorithm 1 (pseudo code of Eclat-Close):

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

```
 1)  start the Eclat algorithm and assign the current node to the variable curr
 2)  {
 3)      if curr.tidset not in hashTable:
 4)          row.tidset ← curr.tidset
 5)          row.eq_class_members ← curr.itemset   // optional
 6)          row.closure ← curr.itemset
 7)          row.support ← cardinality(row.tidset)
 8)          hashTable.add(row)
 9)      else:
10)          row ← hashTable.get(curr.tidset)
11)          row.eq_class_members.add(curr.itemset)   // optional
12)          row.closure ← row.closure ∪ curr.itemset
13)  }
14)  // hashTable is filled; it contains all the frequent equivalence classes
```

5. Experimental Results

In our experiments, we compared *Eclat-Close* with *Apriori-Close* and *Charm* [11]. *Apriori-Close* was presented in Section 3. *Charm* is a very efficient vertical algorithm, also based on *Eclat*. *Charm* reduces the search space to the minimum, i.e. it explores FCIs only. Since *Charm* is a state-of-the-art algorithm, we also compare *Eclat-Close* against it. All three algorithms were implemented in Java. The experiments were carried out on an Intel Quad Core i5-2500 3.3 GHz machine running under Manjaro GNU/Linux with 4 GB RAM. All times reported are real, wall clock times as obtained from the Unix *time* command between input and output. For the experiments we have used the following datasets: T20I6D100K, T25I10D10K, C20D10K, C73D10K and MUSHROOMS. The T20 and T25[3] are sparse datasets, constructed according to the properties of market basket data that are typical weakly correlated data. The C20 and C73 are census datasets from the PUMS sample file, while the MUSHROOMS[4] describes mushrooms characteristics. The last three are highly correlated datasets.

The execution times of the algorithms are illustrated in Table 2. The table also shows the number of FIs and FCIs. *Apriori-Close* finds all FIs and filters FCIs. *Eclat-Close* does the same thing, but in a vertical, depth-first fashion. *Charm* is similar to *Eclat*, but it reduces the search space to FCIs only.

The experiments show that *Eclat-Close* outperforms *Apriori-Close* on all datasets. The difference is especially spectacular on dense datasets. *Eclat-Close* performs very similarly to *Charm*, though it explores a much larger search space. It

[3] `http://www.almaden.ibm.com/software/quest/Resources/`
[4] `http://kdd.ics.uci.edu/`

min_supp	execution time (sec.)			# FIs	# FCIs
	Apriori-Close	Eclat-Close	Charm		
T20I6D100K					
10%	1.30	0.87	0.69	7	7
0.75%	10.15	1.01	2.70	4,710	4,710
0.50%	16.98	1.44	3.64	26,836	26,208
0.25%	43.03	4.62	7.38	155,163	149,217
T25I10D10K					
10%	0.42	0.31	0.27	20	20
0.75%	2.97	0.47	0.79	17,073	7,841
0.50%	16.54	1.04	1.30	302,284	52,033
C20D10K					
30%	7.90	0.31	0.29	5,319	951
20%	19.82	0.35	0.34	20,239	2,519
10%	48.58	0.55	0.47	89,883	8,777
5%	106.75	0.77	0.65	352,611	21,213
C73D10K					
95%	10.09	0.58	0.37	1,007	93
90%	144.64	0.63	0.43	13,463	942
85%	440.65	0.70	0.53	46,575	2,359
MUSHROOMS					
40%	1.00	0.29	0.27	505	124
30%	2.83	0.32	0.28	2,587	425
15%	45.20	0.43	0.34	99,079	2,210
10%	184.84	0.76	0.40	600,817	4,850

Table 2: Response times of Eclat-Close

can be due to the fact that *Charm* needs to apply several tests on an itemset to decide if it is closed. These extra tests give some overhead to *Charm*.

As a conclusion, we can say that *Eclat-Close* performs much better than its levelwise competitor *Apriori-Close*, and it is comparable to *Charm*.

6. Conclusion

In this paper we presented a vertical, depth-first algorithm that finds frequent equivalence classes, i.e. it explores FIs and filters FCIs among them.

When testing the performance of *Eclat-Close* w.r.t. efficient comparable alternatives from the literature, it came out that our algorithm performed surprisingly well for its level of optimization. We tend to see that fact as an indication that any improvement that avoids exploring the entire family of FIs has good chances of becoming the best performing algorithm in its class.

Currently, we only concentrated on FCIs among FIs, but it would be interesting to filter frequent generators as well. Having the generators in an equivalence class

as well, we could produce interesting association rules easily. We plan to extend *Eclat-Close* in this direction.

References

[1] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proceedings of the 20th International Conference 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: Proceedings of the Computational Logic (CL '00). Volume 1861 of LNAI., Springer (2000) 972–986

[3] Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proceedings 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: Proceedings of the 7th International Conference 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] Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proceedings of the 3rd International Conference on Knowledge Discovery in Databases. (August 1997) 283–286

[7] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in knowledge discovery and data mining. American Association for Artificial Intelligence (1996) 307–328

[8] Mannila, H., Toivonen, H.: Levelwise Search and Borders of Theories in Knowledge Discovery. Data Mining and Knowledge Discovery **1**(3) (1997) 241–258

[9] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Closed set based discovery of small covers for association rules. In: Proceedings 15emes Journees Bases de Donnees Avancees (BDA). (1999) 361–381

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

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