ZART: A Multifunctional Itemset Mining Algorithm

Laszlo Szathmary, Amedeo Napoli, Sergei O. Kuznetsov

▶ **To cite this version:**

Laszlo Szathmary, Amedeo Napoli, Sergei O. Kuznetsov. ZART: A Multifunctional Itemset Mining Algorithm. 5th International Conference on Concept Lattices and Their Applications (CLA '07), Oct 2007, Montpellier, France. pp.26–37. inria-00189423

HAL Id: inria-00189423
https://inria.hal.science/inria-00189423

Submitted on 12 Dec 2007

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.

ZART: A Multifunctional Itemset Mining Algorithm

Laszlo Szathmary[1], Amedeo Napoli[1], and Sergei O. Kuznetsov[2]

[1] LORIA UMR 7503, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{szathmar, napoli}@loria.fr
[2] Higher School of Economics, Department of Applied Mathematics
Kirpichnaya 33/5, Moscow 105679, Russia
skuznetsov@hse.ru

Abstract. In this paper, we present and detail a multifunctional itemset mining algorithm called *Zart*, which is based on the *Pascal* algorithm. *Zart* shows a number of additional features and performs the following, usually independent, tasks: identify frequent closed itemsets and associate generators to their closures. This makes *Zart* a complete algorithm for computing classes of itemsets including generators and closed itemsets. These characteristics allow one to extract minimal non-redundant association rules, a useful and lossless representation of association rules. In addition, being based on the *Pascal* algorithm, *Zart* has a rather efficient behavior on weakly and strongly correlated data. Accordingly, *Zart* is at the heart of the Coron platform, which is a domain independent, multi-purposed data mining platform, incorporating a rich collection of data mining algorithms.

1 Introduction

Finding association rules is one of the most important tasks in data mining. Generating valid association rules from frequent itemsets (FIs) often results in a huge number of rules, which limits their usefulness in real life applications. To solve this problem, different concise representations of association rules have been proposed, e.g. generic basis ($\mathcal{GB}$) [1], informative basis ($\mathcal{IB}$) [1], representative rules [2], Duquennes-Guigues basis [3], Luxenburger basis [4], proper basis [5], structural basis [5], etc. A very good comparative study of these bases can be found in [6], where it is stated that a rule representation should be *lossless* (should enable derivation of all valid rules), *sound* (should forbid derivation of rules that are not valid), and *informative* (should allow determination of rules parameters such as support and confidence).

Kryszkiewicz showed in [6] that minimal non-redundant rules[3] ($\mathcal{MNR}$) with the cover operator, and the transitive reduction of minimal non-redundant rules[3] ($\mathcal{RMNR}$) with the cover operator and the confidence transitivity property are lossless, sound, and informative representations of *all* valid association rules.

[3] Defined in Section 2.

From the definitions of $\mathcal{MNR}$ and $\mathcal{RMNR}$ it can be seen that we only need frequent closed itemsets *and* their generators to produce these rules. Frequent itemsets have several condensed representations, e.g. closed itemsets [7–9], generator representation [10], approximate free-sets [11], disjunction-free sets [12], disjunction-free generators [10], generalized disjunction-free generators [13], non-derivable itemsets [14], etc. From these representations, the one which consists of frequent closed itemsets and frequent generators gives rise to a concise set of association rules, which is lossless, sound, and informative [6]. This set of rules, called the set of minimal non-redundant association rules ($\mathcal{MNR}$) [1], is not minimal in general case, but presents a good compromise between its size and time needed to generate it [15].

In [16], Bastide *et al.* presented the *Pascal* algorithm and claimed that $\mathcal{MNR}$ can be extracted with this algorithm. However, to obtain $\mathcal{MNR}$ from the output of *Pascal*, one has to do a lot of computing. First, frequent closed itemsets must also be known. Second, frequent generators must be *associated* to their closures. Here we propose an algorithm called *Zart*, an extension of *Pascal*, which does this computing. Thus, *Zart* allows one to easily construct $\mathcal{MNR}$. Instead of *Pascal*, we might have selected another algorithm. The reason for choosing *Pascal* was as follows: among *levelwise* frequent itemset mining algorithms, it may be the most efficient. This is due to its pattern counting inference mechanism that can significantly reduce the number of expensive database passes. Furthermore, as it was argued in [17], the idea introduced in *Zart* can be generalized, and thus it can be applied to *any* frequent itemset mining algorithm.

The paper is organized as follows. In the next section, we overview the basic concepts and essential definitions. This is followed by the description of the three main features of the *Zart* algorithm. We then present *Zart* and give a running example. Then, the generation of minimal non-redundant association rules is presented. Next, we provide experimental results for comparing the efficiency of *Zart* to *Pascal* and *Apriori*. Finally, we draw conclusions in the last section.

2 Main Definitions

Frequent Itemsets. We consider a set of *objects* $O = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* $A = \{a_1, a_2, \ldots, a_n\}$, and a binary relation $R \subseteq O \times A$, where $R(o, a)$ means that the object o has the attribute a. In formal concept analysis the triple (O, A, R) is called a *formal context* [18]. The Galois connection for (O, A, R) is defined along the lines of [18] in the following way (here $B \subseteq O$, $D \subseteq A$):

$$B' = \{a \in A \mid R(o, a) \text{ for all } o \in B\}, \qquad D' = \{o \in O \mid R(o, a) \text{ for all } a \in D\}.$$

In data mining applications, an element of A is called an *item* and a subset of A is called an *itemset*. Further on, we shall keep to these terms. An itemset of size i is called an i-itemset.[4] We say that an itemset $P \subseteq A$ *belongs* to an object

[4] For instance, {a,b,e} is a 3-itemset. Further on we use separator-free set notations, i.e. *abe* stands for {a,b,e}.

$o \in O$, if $(o, p) \in R$ for all $p \in P$, or $P \subseteq o'$. The *support* of an itemset $P \subseteq A$ indicates the number of objects to which the itemset belongs: $supp(P) = |P'|$. An itemset is *frequent* if its support is not less than a given *minimum support* (denoted by *min_supp*). An itemset P is *closed* if there exists no proper superset with the same support. The closure of an itemset P (denoted by P'') is the largest superset of P with the same support. Naturally, if $P = P''$, then P is a closed itemset. The task of frequent itemset mining consists of generating all (closed) itemsets (with their supports) with supports greater than or equal to a specified *min_supp*.

Two itemsets $P, Q \subseteq A$ are said to be *equivalent* ($P \cong Q$) iff they belong to the same set of objects (i.e. $P' = Q'$). The set of itemsets that are equivalent to an itemset P (P's *equivalence class*) is denoted by $[P] = \{Q \subseteq A \mid P \cong Q\}$. An itemset $P \in [P]$ is called a *generator*[5], if P has no proper subset in $[P]$, i.e. it has no proper subset with the same support. A *frequent generator* is a generator whose support is not less than a given minimum support.

Pattern Counting Inference. The following properties of support and generators were observed in [16] and are usually referred to as properties of *counting inference*.

Property 1. Let P and Q be two itemsets.
(i) $P \cong Q \Rightarrow supp(P) = supp(Q)$
(ii) $P \subseteq Q$ and $(supp(P) = supp(Q)) \Rightarrow P \cong Q$

Property 2. All subsets of a frequent generator are frequent generators.

Property 3. An itemset P is a generator iff $supp(P) \neq min_{p \in P}(supp(P \setminus \{p\}))$.

The last property says that in order to decide whether a candidate set P is a generator, one needs to compare its support to its subsets of size $|P| - 1$. By definition, generators do not admit proper subsets of the same support.

Frequent Association Rules. An association rule is an expression of the form $I_1 \rightarrow I_2$, where I_1 and I_2 are arbitrary itemsets ($I_1, I_2 \subseteq A$), $I_1 \cap I_2 = \emptyset$ and $I_2 \neq \emptyset$. The left side, I_1 is called *antecedent*, the right side, I_2 is called *consequent*. The support of an association rule[6] r is defined as: $supp(r) = supp(I_1 \cup I_2)$. The *confidence* of an association rule r: $I_1 \rightarrow I_2$ is defined as the conditional probability that an object has itemset I_2, given that it has itemset I_1: $conf(r) = supp(I_1 \cup I_2)/supp(I_1)$. An association rule r with $conf(r) = 100\%$ is an *exact* association rule (or implication [18]), otherwise it is an *approximate* association rule. An association rule is *valid* if $supp(r) \geq min_supp$ and $conf(r) \geq min_conf$. The set of valid association rules is denoted by $\mathcal{AR}$.

Now recall the following definitions of bases of association rules:

[5] In the literature these itemsets have various names: key itemsets, minimal generators, free-itemsets, key generators, etc. Further on we will refer to them as "generators".
[6] In this paper we use absolute values, but the support of an association rule r is also often defined as $supp(r) = supp(I_1 \cup I_2)/|O|$.

Definition 1. *Let FCI be the set of frequent closed itemsets. For each frequent closed itemset f, let FG_f denote the set of frequent generators in the equivalence class of f. The generic basis for exact association rules (implications):*

$$\mathcal{GB} = \{r : g \Rightarrow (f \backslash g) \mid f \in FCI \wedge g \in FG_f \wedge g \neq f\}.$$

Definition 2. *Let FCI be the set of frequent closed itemsets and let FG be the set of frequent generators. The informative basis for approximate association rules:*

$$\mathcal{IB} = \{r : g \rightarrow (f \backslash g) \mid f \in FCI \wedge g \in FG \wedge g'' \subset f\}.$$

Definition 3. *Let $\mathcal{IB}$ denote the informative basis for approximate association rules as defined above, and let FCI be the set of frequent closed itemsets. The transitive reduction of $\mathcal{IB}$:*

$$\mathcal{RIB} = \{r : g \rightarrow (f \backslash g) \in \mathcal{IB} \mid g'' \textit{ is a maximal proper subset of } f \textit{ in } FCI\}.$$

Definition 4. *The set of minimal non-redundant rules ($\mathcal{MNR}$) is defined as: $\mathcal{MNR} = \mathcal{GB} \cup \mathcal{IB}$. The transitive reduction of minimal non-redundant rules ($\mathcal{RMNR}$) is defined as: $\mathcal{RMNR} = \mathcal{GB} \cup \mathcal{RIB}$.*

3 Main Features of Zart

Zart has three main features, namely **(1)** pattern counting inference, **(2)** identifying frequent closed itemsets, and **(3)** identifying generators of frequent closed itemsets.

3.1 Pattern Counting Inference in Pascal and Zart

The first part of *Zart* is based on *Pascal*, which employs properties of the counting inference. In levelwise traversal of frequent itemsets, first the smallest elements of an equivalence class are discovered, and these itemsets are exactly the generators. Later, when finding a larger itemset, it is tested if it belongs to an already discovered equivalence class. If it does, the database does not have to be accessed to determine the support of the itemset. This way the expensive database passes and support counts can be constrained to the case of generators only. From some level on, all generators can be found, thus all remaining frequent itemsets and their supports can be inferred without any further database pass.

In Figure 1 (left) we show the output of *Pascal* when executed on dataset $\mathcal{D}$ (Table 4): it finds frequent itemsets and marks frequent generators. Recalling the definitions of $\mathcal{MNR}$ and $\mathcal{RMNR}$, we see that this output is not enough. From our running example, the output of *Zart* is shown in Figure 1 (right). Here one can see the equivalence classes of database $\mathcal{D}$. Only the maximal (frequent closed itemset) and minimal elements (frequent generators) of each equivalence class are indicated. Support values are shown in the top right-hand corner of classes. As can be seen, the output of *Zart* is necessary and sufficient for generating $\mathcal{GB}$, $\mathcal{IB}$, $\mathcal{RIB}$, $\mathcal{MNR}$, and $\mathcal{RMNR}$.

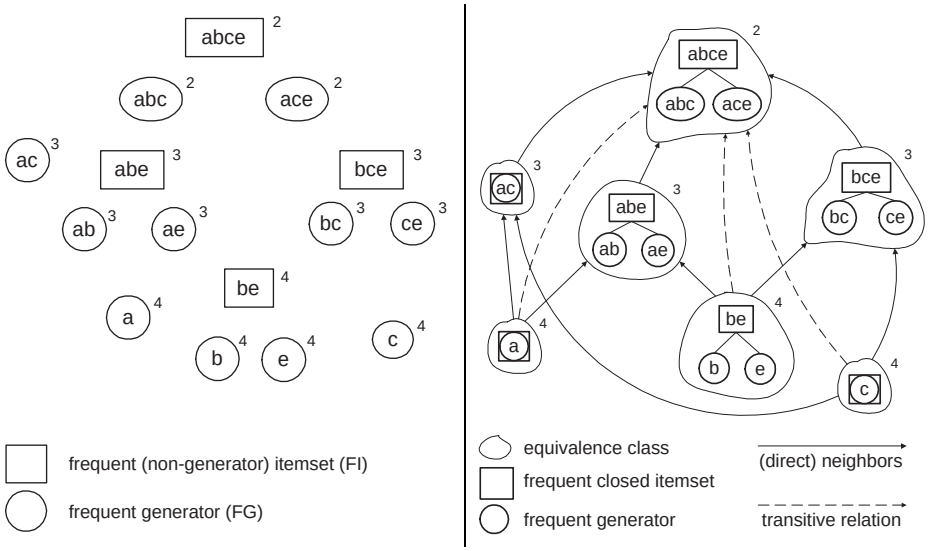

Fig. 1. Result of *Pascal* (left) and *Zart* (right) on $\mathcal{D}$ with $min_supp = 2$ (40%)

3.2 Identifying Closed Itemsets among Frequent Itemsets in Zart

The second part of *Zart* consists in the identification of FCIs among FIs, adapting this idea from *Apriori-Close* [5]. By definition, a closed itemset has no proper superset with the same support. At the i^{th} step all i-itemsets are marked as "closed". At the $(i+1)^{th}$ iteration for each $(i+1)$-itemset we test if it contains an i-itemset with the same support. If so, then the i-itemset is not a closed itemset since it has a proper superset with the same support, thus it is marked as "not closed". When the algorithm terminates with the enumeration of all FIs, itemsets still marked "closed" are the FCIs of the dataset.

3.3 Associating the Generators to their Closures in Zart

Because of the levelwise itemset search, when an FCI is found, all its frequent subsets are already known. This means that its generators are already computed, they only have to be identified. We show that the search space for generators can be narrowed to not closed ones. This is justified by the following properties:

Property 4. A closed itemset cannot be a generator of a larger itemset.

Property 5. The closure of a frequent not closed generator g is the smallest proper superset of g in the set of frequent closed itemsets.

By using these two properties, the algorithm for efficiently finding generators is the following: generators are stored in a list l. At the i^{th} iteration, frequent closed i-itemsets are filtered. For each frequent closed i-itemset z, the following

steps are executed: find the subsets of z in list l, register them as generators of z, and delete them from l. Before passing to the $(i+1)^{th}$ iteration, add the i-itemsets that are not closed generators to list l. Properties 4 and 5 *guarantee* that whenever the subsets of a frequent closed itemset are looked for in list l, only its generators are returned. The returned subsets have the same support as the frequent closed itemset, it does not even have to be tested. Since only the generators are stored in the list, it means that we need to test far fewer elements than the whole set of FIs. Since at step i the size of the largest itemset in list l can be maximum $(i-1)$, we do not find the generators that are identical to their closures. If an FCI has no generator registered, it simply means that its generator is itself. As for the implementation, instead of using a "normal" list for storing generators, the trie data structure is suggested, since it allows a very quick lookup of subsets.

4 The Zart Algorithm

4.1 Pseudo Code

Due to lack of space, we can only give a short overview of the algorithm here. The detailed description of *Zart* can be found in [19]. The main block of the algorithm is given in Table 2. *Zart* uses three different kinds of tables, their descriptions are provided in Table 1. We assume that an itemset is an ordered list of attributes, since we will rely on this in the Zart-Gen function.

Table 1. Tables and table fields used in *Zart*

C_i	potentially frequent candidate i-itemsets fields: (1) itemset, (2) pred_supp, (3) key, (4) support
F_i	frequent i-itemsets fields: (1) itemset, (2) key, (3) support, (4) closed
Z_i	frequent closed i-itemsets fields: (1) itemset, (2) support, (3) gen

itemset	– an arbitrary itemset
pred_supp	– the minimum of the supports of all $(i-1)$-long frequent subsets
key	– is the itemset a generator?
closed	– is the itemset a closed itemset?
gen	– generators of a closed itemset

Zart-Gen function. As input, it gets an F_i table that has a set of frequent itemsets. As output, it returns the C_{i+1} table. The method is the following. It fills the C_{i+1} table with the one-size larger supersets of the itemsets in F_i. The pred_supp values in C_{i+1} are set to the minimum of the supports of all one-size smaller subsets. If a subset is not a generator, then the current itemset is not a generator either, and thus its support is equal to its pred_supp value.

Find-Generators procedure. As input it gets a Z_i table. The method is the following. For each frequent closed itemset z in Z_i, find its proper subsets in the global FG list, register them as generators of z, delete them from FG, and add not closed generators from F_i to FG.

Table 2. Main block of *Zart*

```
1) FG ← { }; // global list of frequent generators
2) fill C1 with 1-itemsets and count their supports;
3) copy frequent itemsets from C1 to F1;
4) mark itemsets in F1 as "closed";
5) mark itemsets in F1 as "key" if their support < |O|;
     //where |O| is the number of objects in the input dataset
6) if there is a full column in the input dataset, then FG ← {∅};
7) i ← 1;
8) loop
9) {
10)      C_{i+1} ← Zart-Gen(F_i);
11)      if C_{i+1} is empty then break from loop;
12)      count the support of "key" itemsets in C_{i+1};
13)      if C_{i+1} has an itemset whose support = pred_supp,
              then mark it as "not key";
14)      copy frequent itemsets to F_{i+1};
15)      if an itemset in F_{i+1} has a subset in F_i with the same
              support, then mark the subset as "not closed";
16)      copy "closed" itemsets from F_i to Z_i;
17)      Find-Generators(Z_i);
18)      i ← i + 1;
19) }
20) copy itemsets from F_i to Z_i;
21) Find-Generators(Z_i);
```

4.2 Running Example

The execution of *Zart* on dataset $\mathcal{D}$ (Table 4, left) is illustrated in Table 3. The algorithm first performs one database scan to count the support values of 1-itemsets. The itemset d is pruned because it is not frequent. At the next iteration, all candidate 2-itemsets are created and their support values are counted. In C_2 there is one itemset with the same support as one of its subsets, thus be is not a generator. Using F_2, the itemsets b and e in F_1 are not closed because they have a proper superset with the same support. The remaining closed itemsets a and c are copied to Z_1 and their generators are determined. In the global list of frequent generators (FG), which is still empty, they have no subsets, which means that both a and c are generators themselves. Not closed generators of F_1 (b and e) are added to the FG list. In C_3, abe and bce turn out to be not generators. Their support values are equal to the support of be (Property 3). By F_3, the itemsets ab, ae, bc, and ce turn out to be not closed. The remaining closed itemsets ac and be are copied to Z_2. The generator of ac is itself, and the generators of be are b and e. These two generators are deleted from FG and ab, ae, bc, and ce are added to FG. The candidate $abce$ is not a generator either, and as there are no more candidate generators in C_4, from this step on no more database scan is needed. In the fifth iteration no new candidate is found and the algorithm breaks out from the main loop. The generators of $abce$ are read from FG. When

Table 3. Execution of *Zart* on dataset $\mathcal{D}$ with $min_supp = 2$ (40%)

DB scan$_1$ →

C_1	pred_supp	key	supp
a			4
b			4
c			4
d			1
e			4

F_1	key	supp	closed
a	yes	4	yes
b	yes	4	~~yes~~
c	yes	4	yes
e	yes	4	~~yes~~

Z_1	supp	gen
a	4	
c	4	

$FG_{before} = \{\}$
$FG_{after} = \{b, e\}$

DB scan$_2$ →

C_2	pred_supp	key	supp
ab	4	yes	3
ac	4	yes	3
ae	4	yes	3
bc	4	yes	3
be	4	~~yes~~	4
ce	4	yes	3

F_2	key	supp	closed
ab	yes	3	~~yes~~
ac	yes	3	yes
ae	yes	3	~~yes~~
bc	yes	3	~~yes~~
be	no	4	yes
ce	yes	3	~~yes~~

Z_2	supp	gen
ac	3	
be	4	$\{b, e\}$

$FG_{before} = \{b, e\}$
$FG_{after} = \{ab, ae, bc, ce\}$

DB scan$_3$ →

C_3	pred_supp	key	supp
abc	3	yes	2
abe	3	~~yes~~	3
ace	3	yes	2
bce	3	~~yes~~	3

F_3	key	supp	closed
abc	yes	2	~~yes~~
abe	no	3	yes
ace	yes	2	~~yes~~
bce	no	3	yes

Z_3	supp	gen
abe	3	$\{ab, ae\}$
bce	3	$\{bc, ce\}$

$FG_{before} = \{ab, ae, bc, ce\}$
$FG_{after} = \{abc, ace\}$

C_4	pred_supp	key	supp
$abce$	2	~~yes~~	2

F_4	key	supp	closed
$abce$	no	2	yes

Z_4	supp	gen
$abce$	2	$\{abc, ace\}$

$FG_{before} = \{abc, ace\}$
$FG_{after} = \{\}$

C_5	pred_supp	key	supp
$\emptyset$			

the algorithm stops, all FIs and all FCIs *with* their generators are determined, as shown in Table 4 (right). In the table the "+" sign means that the frequent itemset is closed. The support values are indicated in parentheses. If *Zart* leaves the generators of a closed itemset empty, it means that the generator is identical to the closed itemset (as this is the case for a, c, and ac in the example).

5 Finding Minimal Non-Redundant Association Rules with Zart

Generating all valid association rules from FIs produces too many rules of which many are redundant. For instance, in dataset $\mathcal{D}$ with $min_supp = 2$ (40%) and $min_conf = 50\%$ no less than 50 rules can be extracted. Considering the small size of the dataset, 5×5, this quantity is huge. How could we find the most interesting rules? How could we avoid redundancy and reduce the number of rules? Minimal non-redundant association rules ($\mathcal{MNR}$) can help us. By Definition 4, an $\mathcal{MNR}$ has the following form: the antecedent is an FG, the

Table 4. A sample dataset $\mathcal{D}$ for the examples (left) and the output of *Zart* (right)

	a	b	c	d	e
1	x	x		x	x
2	x		x		
3	x	x	x		x
4		x	x		x
5	x	x	x		x

All frequent itemsets ($\bigcup_i F_i$)		All frequent closed itemsets with their generators ($\bigcup_i Z_i$)
a (4) +	*be* (4) +	*a* (4); [*a*]
b (4)	*ce* (3)	*c* (4); [*c*]
c (4) +	*abc* (2)	*ac* (3); [*ac*]
e (4)	*abe* (3) +	*be* (4); [*b*, *e*]
ab (3)	*ace* (2)	*abe* (3); [*ab*, *ae*]
ac (3) +	*bce* (3) +	*bce* (3); [*bc*, *ce*]
ae (3)	*abce* (2) +	*abce* (2); [*abc*, *ace*]
bc (3)		

Table 5. Comparing sizes of different sets of association rules

dataset (min_supp)	min_conf	$\mathcal{AR}$ (all valid rules)	$\mathcal{GB}$	$\mathcal{IB}$	$\mathcal{RIB}$	$\mathcal{MNR}$ ($\mathcal{GB} \cup \mathcal{IB}$)	$\mathcal{RMNR}$ ($\mathcal{GB} \cup \mathcal{RIB}$)
$\mathcal{D}$ (40%)	50%	50	8	17	13	25	21
T20I6D100K (0.5%)	90%	752,715	232	721,716	91,422	721,948	91,654
	70%	986,058		951,340	98,097	951,572	98,329
	50%	1,076,555		1,039,343	101,360	1,039,575	101,592
	30%	1,107,258		1,068,371	102,980	1,068,603	103,212
C20D10K (30%)	90%	140,651	967	8,254	2,784	9,221	3,751
	70%	248,105		18,899	3,682	19,866	4,649
	50%	297,741		24,558	3,789	25,525	4,756
	30%	386,252		30,808	4,073	31,775	5,040
C73D10K (90%)	95%	1,606,726	1,368	30,840	5,674	32,208	7,042
	90%	2,053,936		42,234	5,711	43,602	7,079
	85%	2,053,936		42,234	5,711	43,602	7,079
	80%	2,053,936		42,234	5,711	43,602	7,079
Mushrooms (30%)	90%	20,453	544	952	682	1,496	1,226
	70%	45,147		2,961	1,221	3,505	1,765
	50%	64,179		4,682	1,481	5,226	2,025
	30%	78,888		6,571	1,578	7,115	2,122

union of the antecedent and consequent is an FCI, and the antecedent is a proper subset of this FCI. For the generation of such rules, the FCIs *and* their associated generators are needed. Since *Zart* can find both, the output of *Zart* (Table 4, right) can be used directly to generate these rules. For a very quick lookup of frequent closed proper supersets of generators we suggest storing FCIs in a trie.

The algorithm for finding $\mathcal{MNR}$ is the following: for each FG P_1 find its proper supersets P_2 in the set of FCIs. Then add the rule $r : P_1 \rightarrow P_2 \setminus P_1$ to the set of $\mathcal{MNR}$. For instance, using the generator e in Figure 1 (right), three rules can be determined. Rules within an equivalence class form the generic basis ($\mathcal{GB}$), which consists of exact association rules ($e \Rightarrow b$), while rules between equivalence classes are approximate association rules ($e \rightarrow bc$ and $e \rightarrow abc$). For extracting $\mathcal{RMNR}$, the search space for finding frequent closed proper supersets of generators is reduced to equivalence classes that are *direct neighbors*, i.e. transitive relations are eliminated. Thus, for instance, in the previous example

only the first two rules are generated: $e \Rightarrow b$ and $e \rightarrow bc$. A comparative table of the different sets of association rules, that can be extracted easily using the output of *Zart*, is shown in Table 5.[7] In sparse datasets, like T20I6D100K, the number of $\mathcal{MNR}$ is not much less than the number of $\mathcal{AR}$. However, in dense, highly correlated datasets, like C20D10K or MUSHROOMS, the difference is significant. $\mathcal{RMNR}$ always represents much less rules than $\mathcal{AR}$, in sparse and dense datasets too.

6 Experimental Results

We evaluated *Zart* against *Apriori* and *Pascal*. We have implemented these algorithms in Java using the same data structures, and they are all part of the CORON data mining platform [20]. The experiments were carried out on an Intel Pentium IV 2.4 GHz machine running Debian GNU/Linux operating system with 512 MB 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, C20D10K, and MUSHROOMS. The T20I6D100K[8] is a sparse dataset, constructed according to the properties of market basket data that are typical weakly correlated data. The C20D10K is a census dataset from the PUMS sample file, while the MUSHROOMS[9] describes mushrooms characteristics. The last two are highly correlated datasets. It has been shown that weakly correlated data, such as synthetic data, constitute easy cases for the algorithms that extract frequent itemsets, since few itemsets are frequent. For such data, all algorithms give similar response times. On the contrary, dense and highly-correlated data constitute far more difficult cases for the extraction due to large differences between the number of frequent and frequent closed itemsets. Such data represent a huge part of real-life datasets. Response times for the datasets are presented numerically in Table 6.

6.1 Weakly Correlated Data

The T20I6D100K synthetic dataset mimics market basket data that are typical sparse, weakly correlated data. In this dataset, the number of FIs is small and nearly all FIs are generators. *Apriori*, *Pascal*, and *Zart* behave identically. As we can see in T20I6D100K, above 0.75% minimum support all frequent itemsets are closed and generators at the same time. It means that each equivalence class has one element only. Because of this, *Zart* and *Pascal* cannot use the advantage of pattern counting inference and they work exactly like *Apriori*.

6.2 Strongly Correlated Data

In datasets C20D10K and MUSHROOMS, the number of FGs is much less than the total number of FIs. Hence, using pattern counting inference, *Zart* has to

[7] Note that in the case of $\mathcal{GB}$, by definition, minimum confidence is 100%.
[8] http://www.almaden.ibm.com/software/quest/Resources/
[9] http://kdd.ics.uci.edu/

Table 6. Response times of *Zart* and other statistics (number of FIs, number of FCIs, number of FGs, proportion of the number of FCIs to the number of FIs, proportion of the number of FGs to the number of FIs)

min_supp	execution time (sec.)			# FIs	# FCIs	# FGs	$\frac{\#FCIs}{\#FIs}$	$\frac{\#FGs}{\#FIs}$
	Apriori	Pascal	Zart					
T20I6D100K								
2%	72.67	71.15	71.16	378	378	378	100.00%	100.00%
1%	107.63	106.24	107.69	1,534	1,534	1,534	100.00%	100.00%
0.75%	134.49	132.00	133.00	4,710	4,710	4,710	100.00%	100.00%
0.5%	236.10	228.37	230.17	26,836	26,208	26,305	97.66%	98.02%
0.25%	581.11	562.47	577.69	155,163	149,217	149,447	96.17%	96.32%
C20D10K								
50%	61.18	16.68	17.94	1,823	456	456	25.01%	25.01%
40%	71.60	19.10	19.22	2,175	544	544	25.01%	25.01%
30%	123.57	26.74	26.88	5,319	951	967	17.88%	18.18%
20%	334.87	53.28	54.13	20,239	2,519	2,671	12.45%	13.20%
10%	844.44	110.78	118.09	89,883	8,777	9,331	9.76%	10.38%
MUSHROOMS								
60%	3.10	2.04	2.05	51	19	21	37.25%	41.18%
50%	6.03	3.13	3.13	163	45	53	27.61%	32.52%
40%	13.93	6.00	6.03	505	124	153	24.55%	30.30%
30%	46.18	12.79	12.84	2,587	425	544	16.43%	21.03%
20%	554.95	30.30	34.88	53,337	1,169	1,704	2.19%	3.19%

perform much fewer support counts than *Apriori*. We can observe in all cases that the execution times of *Zart* and *Pascal* are almost identical: adding the FCI derivation *and* the identification of their generators to the FI discovery does not induce serious additional computation time. *Apriori* is very efficient on sparse datasets, but on strongly correlated data the other two algorithms perform much better.

7 Conclusion and Future Work

In this paper we presented a multifunctional itemset mining algorithm called *Zart*, which is a refinement of *Pascal*. As an addition, it can identify frequent closed itemsets, and it can associate generators to their closure. We showed that these extra features are essential for the generation of minimal non-redundant association rules. Experimental results show that *Zart* gives almost equivalent response times to *Pascal* on both weakly and strongly correlated data.

An interesting question is the following: can the idea of *Zart* be generalized and used for *any* arbitrary frequent itemset mining algorithm, be it either breadth-first or depth-first? Could we somehow extend these algorithms in a universal way to produce such results that can be used directly to generate not only all valid association rules, but minimal non-redundant association rules too? Our answer is positive [17], but detailed study of this will be subject of a next paper.

References

1. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining minimal non-redundant association rules using frequent closed itemsets. In Lloyd, J.*et al.*., ed.: Proc. of CL'00. Volume 1861 of LNAI., Springer (2000) 972–986
2. Kryszkiewicz, M.: Representative association rules. In: Proc. of PAKDD '98, London, UK, Springer-Verlag (1998) 198–209
3. Guigues, J.L., Duquenne, V.: Familles minimales d'implications informatives résultant d'un tableau de données binaires. Math. et Sci. Hum. **95** (1986) 5–18
4. Luxenburger, M.: Implications partielles dans un contexte. Mathématiques, Informatique et Sciences Humaines **113** (1991) 35–55
5. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Closed set based discovery of small covers for association rules. In: Proc. of BDA '99. (1999) 361–381
6. Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery. (2002) 92–109
7. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. Inf. Syst. **24**(1) (1999) 25–46
8. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. LNCS **1540** (1999) 398–416
9. Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: Proc. of SDM '02. (2002) 33–43
10. Kryszkiewicz, M.: Concise representation of frequent patterns based on disjunction-free generators. In: Proc. of ICDM '01, Washington, DC, USA, IEEE Computer Society (2001) 305–312
11. Boulicaut, J.F., Bykowski, A., Rigotti, C.: Approximation of frequency queries by means of free-sets. In: Proc. of PKDD '00, Lyon, France, Springer Berlin / Heidelberg (2000) 75–85
12. Bykowski, A., Rigotti, C.: A condensed representation to find frequent patterns. In: Proc. of PODS '01, ACM Press (2001) 267–273
13. Kryszkiewicz, M., Gajek, M.: Why to apply generalized disjunction-free generators representation of frequent patterns? In: Proc. of ISMIS 2002, Lyon, France, Springer-Verlag Berlin / Heidelberg (2002) 383–392
14. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In: Proc. of PKDD '02, London, UK, Springer-Verlag (2002) 74–85
15. Pasquier, N.: Mining association rules using formal concept analysis. In: Proc. of ICCS '00, Shaker-Verlag (2000) 259–264
16. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining frequent patterns with counting inference. SIGKDD Explor. Newsl. **2**(2) (2000) 66–75
17. Szathmary, L.: Symbolic Data Mining Methods with the Coron Platform (Méthodes symboliques de fouille de données avec la plate-forme Coron). PhD in Computer Sciences, Univ. Henri Poincaré Nancy 1, France (2006)
18. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin/Heidelberg (1999)
19. Szathmary, L., Napoli, A., Kuznetsov, S.O.: ZART: A Multifunctional Itemset Mining Algorithm. LORIA Research Report 00001271 (2005)
20. Szathmary, L., Napoli, A.: CORON: A Framework for Levelwise Itemset Mining Algorithms. In: Suppl. Proc. of ICFCA '05, Lens, France. (2005) 110–113