Mining disjunctive minimal generators with TitanicOR

Renato Vimieiro[a,*], Pablo Moscato[a]

[a] *Centre for Bioinformatics, Biomarker Discovery & Information-Based Medicine*
The University of Newcastle, Callaghan, 2308, NSW, Australia
Tel.: +61 2 49216056, Fax: +61 2 49216929

Abstract

Disjunctive minimal generators were proposed by Zhao et al. (2006). They defined disjunctive closed itemsets and disjunctive minimal generators through the disjunctive support function. We prove that the disjunctive support function is *compatible* with the closure operator presented by Zhao et al. (2006). Such compatibility allows us to adapt the original version of the Titanic algorithm, proposed by Stumme et al. (2002) to mine iceberg concept lattices and closed itemsets, to mine disjunctive minimal generators. We present TitanicOR, a new breadth-first algorithm for mining disjunctive minimal generators. We evaluate the performance of our method with both synthetic and real data sets and compare TitanicOR's performance with the performance of BLOSOM (Zhao et al., 2006), the state of the art method and sole algorithm available prior to TitanicOR for mining disjunctive minimal generators. We show that TitanicOR's breadth-first approach is up to two orders of magnitude faster than BLOSOM's depth-first approach.

Keywords: TitanicOR, BLOSOM, minimal generators, closed itemsets, disjunctions, Boolean expressions, frequent pattern mining

1. Introduction

We present TitanicOR, an algorithm to mine disjunctive minimal generators. TitanicOR is an adaptation of the Titanic algorithm (Stumme et al., 2002) originally proposed to mine either conjunctive minimal generators or iceberg concept lattices, i.e., (frequent) closed itemsets. Minimal generators play an important role in the task of pattern mining since they represent the most general description of a set of objects. The task of mining disjunctive minimal generators was introduced by Zhao et al. (2006) as a part of BLOSOM, a framework to mine Boolean expressions. While conjunctive minimal generators are the minimal sets of attributes for which each attribute occurs in *all* objects in a set of objects, disjunctive minimal generators are the minimal sets of attributes for which each attribute occurs in *at least one* object of a set of objects.

Frequent pattern mining is one of the steps of the *Association analysis* (Agrawal et al., 1993). It consists of enumerating all the possible combinations of attributes in a database and retrieving those sets of attributes (itemsets) that surpass a user specified minimum frequency threshold. The frequency of an itemset is the number of objects where the attributes of the set co-occur in the database. Such a definition of the problem leads to a conjunctive interpretation of the pattern since an object counts for the frequency of a pattern only if it contains *all* the items in the pattern.

Despite being a computationally hard problem in the worst case scenario, there are several algorithms to mine all frequent itemsets from databases that have been extensively applied to real world problems. The extraction of all frequent itemsets from a transactional data, albeit feasible, can result in a very large number of sets in practice, thus, the problem is just shifted from analyzing the raw data to analyzing a huge number of frequent itemsets. Therefore, concise representations for the set of all frequent itemsets that still preserves the same information as the original set are required; two of such representations are: *frequent closed itemsets*, and *minimal generators*.

*Corresponding author
Email addresses: Renato.Vimieiro@newcastle.edu.au (Renato Vimieiro), Pablo.Moscato@newcastle.edu.au (Pablo Moscato)

The frequent closed itemset representation provides a minimal representation for the set of all frequent itemsets without losing frequency information. The frequency of an itemset represents the size of the set of objects associated with that itemset. Different itemsets can be associated with the same set of objects. The collection of all sets of objects associated with frequent itemsets induces a partition of the set of frequent itemsets in which each equivalence class contains all frequent itemsets associated with the same set of objects. A *closed itemset* is the maximal itemset within an equivalence class.

Minimal generators are related to closed itemsets. While the latter represent the maximal itemsets within an equivalence class, *minimal generators* represent the minimal itemsets within an equivalence class. Regarding a set of objects, minimal generators are its most general representation whereas closed itemsets are its most specific.

There is a growing interest in other interpretations of those patterns such as disjunctive, and negative interpretations. There are situations where conjunctive interpretation may not be as suitable as the others. For example, it might be interesting for a supermarket manager to know that the purchase of at least one of the items in a set of items often implies in the purchase of another set of items. Or, a biologist might be interested to know that, although there are some proteins whose individual absence, or under-expression may not be associated with the development of a disease, their union may totally describe that condition (Sahoo et al., 2008; Varadan and Anastassiou, 2006). More recently, Hamrouni et al. (2009) demonstrated that both the disjunctive search space may be used to both mine conjunctive itemsets and define another concise representation for the set of itemsets. So, given the theoretical and practical usefulness of the disjunctive itemsets, several approaches were developed to extend the original definition of patterns increasing their expressive power (Srikant and Agrawal, 1995; Savasere et al., 1998; Nanavati et al., 2001; Zhao et al., 2006; Hamrouni et al., 2009; Thummalapenta and Xie, 2011).

Motivated by the necessity of expanding the expressive power of itemsets, followed by the lack of algorithms to perform this task as pointed out by Hamrouni et al. (2009), we propose in this work, TitanicOR, an extension of the Pascal/Titanic (Bastide et al., 2000; Stumme et al., 2002) algorithm to mine disjunctive minimal generators. We use the closure operator proposed by Zhao et al. (2006) with whom we compare our method. Since the original Titanic algorithm requires a *compatible weight function* to compute minimal generators, we demonstrate that the disjunctive support function (Zhao et al., 2006) is a compatible weight function with the considered closure operator. We also believe that the breadth-first approach (used by our algorithm) is more suitable to mine minimal generators than the depth-first approach used by BLOSOM (Zhao et al., 2006). Therefore, to support our hypothesis, we conduct tests using both synthetic and real data to evaluate the performance of the algorithms with data sets with different densities, containing different number of attributes, and containing different number of objects; we also evaluate the impact of the interestingness parameters, such as minimum support, maximum support, and maximum size of itemsets, on the algorithms.

We organize the paper as follows: in the next section, we discuss some preliminary concepts; in Section 3, we present *TitanicOR* our version of Titanic (Stumme et al., 2002) to mine disjunctive minimal generators; then, we present some experimental results in Section 4; Section 5 contains some related work; and, finally, we present our conclusions in Section 6.

2. Foundations

Let S be a set of sample IDs, I a set of items and $R \subseteq S \times I$ be a binary (incidence) relation, where $(s, i) \in R$ shall be read as 'the sample s has the item i'. A *data set* $\mathcal{D}$ is the triplet (S, I, R). Table 1 depicts an example data set for which $S = \{fish, toad, human, monkey, owl, shark\}$, $I = \{aquatic, terrestrial, branchia, lung, feather, hair, mammal, reason\}$ and R is the set of all cells that are marked with '×'.

A subset Y of I is called an *itemset*. Let $f : \mathcal{P}(I) \rightarrow \mathcal{P}(S)$, $f(Y) = \{s \in S \mid \exists i \in Y\ [(s, i) \in R]\}$ denote the set of samples associated with the items in Y (Zhao et al., 2006) – we use $\mathcal{P}(X)$ to denote $\{W \subseteq X\}$. We call $|f(Y)|$ the *(disjunctive) support* of an itemset Y. We say that an itemset is *frequent* if its support is at least as great as a user specified *minimum support* threshold. Table 2 presents a list of itemsets, their associated set of samples in regard to the function f , their disjunctive support and if the itemset is a frequent itemset or not considering a minimum support threshold of three. We represent items and samples with their initials and drop the braces and commas in the set notation, then, for example, the itemset $\{aquatic, lung\}$ is represented by al.

The same way the set of samples associated with a set of items is described by the function f, Zhao et al. (2006) defined, for a set of samples $X \subseteq S$, $g : \mathcal{P}(S) \rightarrow \mathcal{P}(I)$, $g(X) = \{i \in I \mid f(\{i\}) \subseteq X\}$ as the largest set of items describing

Table 1: An example data set

Animals	aquatic	terrestrial	branchia	lung	feather	hair	mammal	reason
fish	×		×					
toad	×	×	×	×				
human		×		×		×	×	×
monkey		×		×		×	×	
owl		×		×	×			
shark	×		×					

Table 2: Example of itemsets from the data set in Table 1

Itemset	Associated set of samples	Supp.	Frequent
r	h	1	no
f	o	1	no
fr	oh	2	no
m	hm	2	no
hm	hm	2	no
tmr	$thmo$	4	yes
t	$thmo$	4	yes
al	$fthmos$	6	yes

the samples of X. For example, considering the set of samples fts, $g(fts) = ab$. For some set of samples, however, it may be the case that there is no such combination of items capable of exclusively describing the set of samples, then the result of the function g is the empty set as it happens with the set fo. In this case, there is no item in Table 1 such that the set of samples associated with it is a subset of fo.

Let $\langle \mathcal{P}(I), \subseteq \rangle$ be the set of all itemsets partially ordered by set inclusion and let $\langle \mathcal{P}(S), \subseteq \rangle$ be the collection of all sets of samples partially ordered by set inclusion. We will denote both the partial orders and the power sets by $\mathcal{P}(I)$ and $\mathcal{P}(S)$ and it will be clear from the context if we are denoting the partial order or the power set. Zhao (2006) proved that the pair (f, g) forms a *Galois connection* over $\mathcal{P}(I)$ and $\mathcal{P}(S)$. Thus, the composite function $g \circ f$ forms a closure operator on $\mathcal{P}(I)$. We will assume that $h = g \circ f$.

According to Ganter and Wille (1997), any closure operator gives rise to a collection of closed sets and vice versa. Thus, we can use a closure operator to determine, or in our case, define a closed set. Let $X \in \mathcal{P}(I)$ be an itemset. We say that X is a *closed itemset* if $X = h(X)$.

The closure operator also induces a partition in the set of all possible itemsets; for an itemset $X \subseteq I$, the equivalence class of X is defined as $[X] = \{Y \subseteq I \mid h(Y) = h(X)\}$ (i.e. the set of itemsets that have the same closure as X). We call the minimal elements of $[X]$ (with respect to $\subseteq$) the *minimal generators* of $h(X)$ (Bastide et al., 2000; Stumme et al., 2002).

Stumme et al. (2002) showed how a *compatible weight function* can be used to compute the closed sets and their minimal generators from I. They defined a *weight function* as a mapping from I to a totally ordered set $\langle P, \leq \rangle$, and proved that a weight function w is *compatible* with a closure operator c if w presents the following properties.

Definition 1 (Stumme et al. (2002)). Let $X, Y \subseteq I$ be two itemsets, $\langle P, \leq \rangle$ a totally ordered set, $c : \mathcal{P}(I) \to \mathcal{P}(I)$ a closure operator, and $w : \mathcal{P}(I) \to \langle P, \leq \rangle$ an arbitrary function. We say that w is a *compatible weight function* with c if:

1. $X \subseteq Y \to w(X) \leq w(Y)$;
2. $c(X) = c(Y) \to w(X) = w(Y)$;
3. $X \subseteq Y \wedge w(X) = w(Y) \to c(X) = c(Y)$

Stumme et al. (2002) showed that the traditional conjunctive support function is compatible with the closure operator defined by Ganter and Wille (1997) for conjunctive closed itemsets. Here we show that the disjunctive support function $|f|$ is compatible with the closure operator h proposed by Zhao et al. (2006).

Proposition 1. *Let (S, I, R) be an arbitrary data set and $A, A_1, A_2 \subseteq I$ arbitrary itemsets. The functions f, g and h present the following properties.*

1. $A \subseteq h(A)$;
2. $A_1 \subseteq A_2 \rightarrow f(A_1) \subseteq f(A_2)$;
3. $f(A) = f(h(A))$.

Proof.

1. By definition, since h is a closure operator;
2. Suppose that $A_1 \subseteq A_2$. Let $s \in S$ be an arbitrary sample such that $s \in f(A_1)$. Since $s \in f(A_1)$, by definition, there is $i \in A_1$ such that $(s, i) \in R$. So, let $i \in A_1$ be an item such that $(s, i) \in R$. Given that $A_1 \subseteq A_2$, $i \in A_2$ and, thus, $s \in f(A_2)$. Therefore, if $A_1 \subseteq A_2$, then $f(A_1) \subseteq f(A_2)$;
3. From 1 and 2, it follows that $f(A) \subseteq f(h(A))$. Now, let $s \in f(h(A))$. Suppose that $s \notin f(A)$. So, $\forall i \in A (s, i) \notin R$. Since $A \subseteq h(A)$, we have that $s \notin f(h(A))$, contradicting the assumption that $s \in f(h(A))$. Hence, $s \in f(A)$. Since s is an arbitrary sample, it follows that $f(h(A)) \subseteq f(A)$. Therefore, $f(A) = f(h(A))$.

□

Proposition 2. *The disjunctive support function $|f|$ is compatible with the closure operator h.*

Proof. To prove that $|f|$ is compatible with h, we need to show that the properties discussed in Definition 1 hold. In the proof below, let (S, I, R) be an arbitrary data set, and $X, Y \subseteq I$ be arbitrary itemsets.

1. $X \subseteq Y \rightarrow |f(X)| \leq |f(Y)|$:
   Suppose that $X \subseteq Y$. Then it follows that $f(X) \subseteq f(Y)$, by Proposition 1 item 2. Consequently $|f(X)| \leq |f(Y)|$. Therefore, if $X \subseteq Y$, then $|f(X)| \leq |f(Y)|$.
2. $h(X) = h(Y) \rightarrow |f(X)| = |f(Y)|$:
   Suppose that $h(X) = h(Y)$, then, $f(h(X)) = f(h(Y))$. Proposition 1.3 states that $f(X) = f(h(X))$ and $f(Y) = f(h(Y))$, so $f(X) = f(h(X)) = f(h(Y)) = f(Y)$. Thus, $|f(X)| = |f(Y)|$ and therefore, if $h(X) = h(Y)$, then $|f(X)| = |f(Y)|$.
3. $X \subseteq Y \wedge |f(X)| = |f(Y)| \rightarrow h(X) = h(Y)$:
   Suppose that $X \subseteq Y$, and $|f(X)| = |f(Y)|$. Since h is a closure operator, by definition, it follows that $h(X) \subseteq h(Y)$. Now, let $i \in h(Y)$ be an arbitrary item. By definition of h, $f(\{i\}) \subseteq f(Y)$. Since $X \subseteq Y$, it follows from Proposition 1.2 that $f(X) \subseteq f(Y)$. As we assume that $|f(X)| = |f(Y)|$, we have $f(X) = f(Y)$ and, thus, $f(\{i\}) \subseteq f(X)$. So $i \in g(f(X))$ by definition of g, or, in other words, $i \in h(X)$. As i is an arbitrary element $h(Y) \subseteq h(X)$. We conclude then that $X \subseteq Y \wedge |f(X)| = |f(Y)| \rightarrow h(X) = h(Y)$.

□

Proposition 3 (Adapted from (Stumme et al., 2002, Proposition 8))**.** *Let $X \subseteq I$.*

1. *For some $i \in X$, $X \in [X \setminus \{i\}]$ if and only if $|f(X)| = |f(X \setminus \{i\})|$.*
2. *X is a minimal generator if and only if $|f(X)| \neq max\{|f(X \setminus \{i\})| \mid i \in X\}$*

Proof. The proof follows from items 2 and 3 in Proposition 2. We refer to (Stumme et al., 2002, Proposition 8) for more details.

□

Proposition 4 (Equivalent to (Stumme et al., 2002, Proposition 5))**.** *If an itemset Y is not a minimal generator, then every superset of X is not a minimal generator as well.*

Proposition 3 is useful for pruning candidate itemsets as we discuss in Section 3.

Since we proved that the disjunctive support function is compatible with the closure operator for disjunctive itemsets and showed how to prune uninteresting candidates we are able, at this point, to adapt the original Titanic algorithm (Stumme et al., 2002), and create TitanicOR to mine disjunctive minimal generators. However, we first present another proposition that helps to speed up our method.

Proposition 5. *Let* (S, I, R) *be an arbitrary data set, and X and Y be two itemsets.* $f(X \cup Y) = f(X) \cup f(Y)$.

Corollary 1. *Let X and Y be two itemsets.* $|f(X \cup Y)| = |f(X) \cup f(Y)|$.

As we will see later, we can use Corollary 1 to speed up the computation of the itemsets' supports.

Now that we presented all the fundamentals for our method, we introduce it in the next section.

3. TitanicOR

The TitanicOR algorithm is an adaptation of the original conjunctive version of Titanic proposed by Stumme et al. (2002) that computes disjunctive minimal generators. TitanicOR operates in a "level-wise" manner like Apriori (Agrawal and Srikant, 1994); it first finds all itemsets of size k before it starts computing itemsets of size $k+1$. It also uses pruning strategies to reduce the search space; it considers both the equivalence classes induced by the closure operator and the properties of weight functions to prune off candidates. Since we are interested only in the minimal generators of each equivalence class, we can use Proposition 3 to prune invalid candidates. Algorithm 1 presents the pseudo-code for TitanicOR while Table 3 presents a list of notations used in Algorithm 1.

1: TitanicOR

Input: A data set $\mathcal{D} = (S, I, R)$

Output: K the set of minimal generators in $\mathcal{D}$

```
1  begin
2      ∅.s = 0;
3      K[0] = {∅};
4      k = 1;
5      C = ∅;
6      foreach i ∈ I do
7          {i}.ps = 0;
8          {i}.t = {t ∈ S | (t, i) ∈ R};
9          {i}.s = |{i}.t|;
10         if {i}.s ≤ maxsup and {i}.s ≥ minsup then
11             C = C ∪ { {i} };
12         end
13     end
14     while (K[k] = {X ∈ C | X.s ≠ X.ps}) ≠ ∅ do
15         k++;
16         C = TitanicORGen(K[k − 1]);
17     end
18     return K
19 end
```

TitanicOR initially considers the empty set as a minimal generator for the empty set of samples (lines 1 and 2). Next, it considers each item in the data set as a candidate minimal generator (lines 6–13). The algorithm associates to each itemset three variables, ps, s, t, storing, respectively, the parent weight (maximum support of an immediate subset), its own weight (its support) and the set of samples associated with it (lines 7–9). After the initialization process,

Table 3: List of notations used in Algorithm 1 describing the TitanicOR algorithm

$X.s$	$X.s$ stores the support of an itemset X
$X.ps$	$X.ps$ stores the support of the "parent" (immediate subset) of an itemset X
$X.t$	$X.t$ stores the set of transactions associated with an itemset X
C	stores the set of candidate minimal generators
K	stores the set of minimal generators

Function `TitanicORGen`

Input: $K[k-1]$ the set of minimal generators of size $k-1$
Output: C a set of candidate minimal generators of size k

```
20 begin
21     C = ∅;
22     forall X, Y ∈ K[k − 1] such that (∀_{i<k−1} X_i = Y_i) ∧ X_{k−1} < Y_{k−1} do
23         W = X ∪ Y;
24         W.t = X.t ∪ Y.t;
25         W.s = |W.t|;
26         W.ps = 0;
27         include = true;
28         if W.s > maxsup or W.s < minsup then
29             skip W;
30         end
31         forall S ∈ {G ⊂ W | |G| = k − 1} do
32             if S ∉ K[k − 1] then
33                 include = false;
34                 exit forall;
35             end
36             W.ps = max(W.ps, S.s);
37         end
38         if include = true then
39             C = C ∪ {W}
40         end
41     end
42     return C
43 end
```

TitanicOR enters a loop to compute minimal generators level-wise (lines 14–17). It determines which of the candidates are minimal generators using Proposition 3, and then, it computes the next candidates using the TitanicORGen function.

The TitanicORGen algorithm is a modification of TitanicGen (Stumme et al., 2002). To generate the new candidates of size k, it assumes a strict total order over the items of I, then it joins two minimal generators of size $k-1$ that share the first $k-2$ items (line 3–22); it restricts the $(k-1)th$ item of the first set to be lower than the $(k-1)th$ item of the second to avoid the same candidate being generated twice. The set of samples associated with the new candidate is computed using Proposition 5 (line 5). We propose this adaptation of the original version of TitanicGen to speed up the computation of itemsets' weights, requiring less data set passes; assuming that the sets of samples associated with the itemsets fit the in main memory. We also consider the set of samples as a bit-array, so the unions can be computed faster, especially when the algorithm is coded and executed on *single-instruction, multiple-data* (SIMD) architectures. Hence computing the disjunctive support of an itemset consists of counting the number of bits set to one in the bit-array of the set of samples, which can be done in constant time (line 6). Next, the algorithm checks if the new candidate passes the minimum support (*minsup*) and the maximum support (*maxsup*) thresholds (lines 9–11). After that, it prunes off candidates that are not minimal (lines 12–18) based on Proposition 4. At the same time TitanicORGen verifies whether an immediate subset of a candidate is a minimal generator, it computes the parent weight of a candidate (line 17) to be used in TitanicOR to prune invalid candidates using Proposition 3. Finally, if a candidate passes all the tests, the algorithm includes it to the list of candidates C. We present an example of TitanicOR and TitanicORGen in practice in Example 1.

Example 1. *In this example, we show how TitanicOR extracts minimal disjunctive generators from a data set. We use the data set presented in Table 1 as the input data set, and we consider minimum and maximum support thresholds of two and five respectively (*$minsup = 2$*,* $maxsup = 5$*).*

Initially, we have $K[0] = \{\emptyset\}, k = 1,$ *and* $C = \emptyset$ *(lines 2–5). After the initialization, the algorithm considers the singletons built from the items in the data set as the first candidates (lines 6–13). We present the results of this process in Table 4.*

Table 4: This table presents the output candidates of TitanicOR after processing the singletons from the items of the data set Table 1 (lines 6–13, Algorithm 1). The rows in light-gray represent the singletons that were discarded for failing the support constraints.

W	$W.t$	$W.s$	$W.ps$	$W \in C$?
a	fts	3	0	yes
t	thmo	4	0	yes
b	fts	3	0	yes
l	thmo	4	0	yes
f	o	1	0	no
h	hm	2	0	yes
m	hm	2	0	yes
r	h	1	0	no

After processing the singletons, the algorithm obtains the candidate list $C = \{a, t, b, l, h, m\}$*, and the set of minimal generators of size one can be computed. The minimal generators are those whose parent weight is different from their own weight,* i.e. *an itemset X is a minimal generator if its support (X.s) is different from the maximum support of all its immediate proper subsets (X.ps). In that case, the list of minimal generators of size one is* $K[1] = \{a, t, b, l, h, m\}$.

Since the list of minimal generators of size one is not empty, the algorithm proceeds to generate new candidates of size two incrementing the value of k and invoking TitanicORGen function (lines 15–16, Algorithm 1). TitanicORGen generates candidates of size two by merging two minimal generators of size one; Table 5 shows all the possible candidates considered by TitanicORGen, and whether they are valid candidates or not.

Table 5: This table presents the list of all possible candidates of size two considered by TitanicORGen (lines 3–22, Algorithm TitanicORGen). The rows in light-gray represent the candidates that were discarded for either failing the support constraints or the requirements of Proposition 4.

W	$W.t$	$W.s$	$W.ps$	$W \in C$?
at	fthmos	6	4	no
ab	fts	3	3	yes
al	fthmos	6	4	no
ah	fthms	5	3	yes
am	fthms	5	3	yes
tb	fthmos	6	4	no
tl	thmo	4	4	yes
th	thmo	4	4	yes
tm	thmo	4	4	yes
bl	fthmos	6	4	no
bh	fthms	5	3	yes
bm	fthms	5	3	yes
lh	thmo	4	4	yes
lm	thmo	4	4	yes
hm	hm	2	2	yes

After the new candidates are generated by TitanicORGen, TitanicOR evaluates which are minimal generators indeed. Again, the minimal generators are the candidates whose support is different from the parent weight. Checking the list of candidates generated by TitanicORGen (Table 5), we get the following list of minimal generators of size two $K[2] = \{ah, am, bh, bm\}$.

As $K[2]$ *is not empty TitanicOR proceeds to compute new candidates. TitanicORGen is invoked again, and it considers all the sets presented in Table 6 as possible candidates. None of the possible sets is a valid candidate since one of their immediate subsets is not a minimal generator and so they fail the test in line 13 of TitanicORGen function (Proposition 4).*

Table 6: This table presents the list of all possible candidates of size three considered by TitanicORGen (lines 3–22, Algorithm TitanicORGen). The rows in light-gray represent the candidates that were discarded for failing the requirements of Proposition 4.

W	$W.t$	$W.s$	$W.ps$	$W \in C$?
ahm	fthms	5	5	no
bhm	fthms	5	5	no

Since TitanicORGen returns an empty list of candidates, there is no minimal generator of size three. Therefore, TitanicOR stops the loop (lines 14–17) and returns the list of minimal generators $K = \{\emptyset, a, t, b, l, h, m, ah, am, bh, bm\}$.

4. Experimental results

TitanicOR was implemented in C++ and compiled with GCC-3.4.6 (code is available upon request from the authors). The experiments were run on a Red-Hat 3.4.6-11 system on a Intel Xeon model E5405 2.00 GHz quad-core

Table 7: Default parameter values for generating synthetic data sets for comparing the performances of TitanicOR and BLOSOM

Parameter	Value
# of samples	300
# of items	50
density	0.5
minsup	0
maxsup	∞
max_item	4

64-bit processor with 32 GB memory. We perform experiments similar to those done by Zhao (2006) to investigate TitanicOR's behavior regarding different data sets, and also compare its performance with the performance of BLOSOM (we use the authors' implementation, available in http://www.cs.rpi.edu/~zaki). We use both synthetic and real data sets. It is important to mention at this point that BLOSOM requires a vertical representation of the data set, the implementation we obtained creates a binary file containing such representation, therefore, we run BLOSOM twice for each experiment: first to create such representation; and second to measure the CPU-times for it.

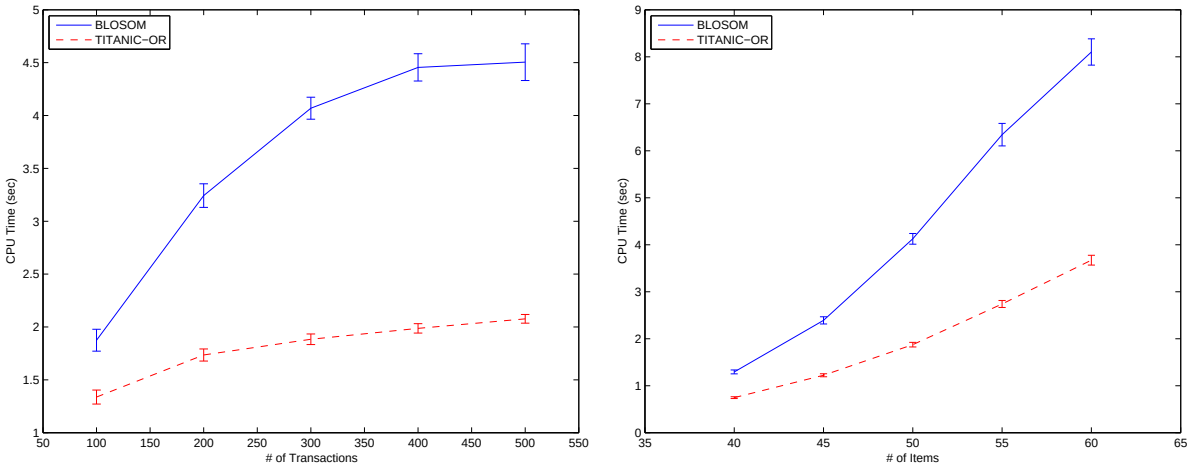
The synthetic data sets were generated using the IBM data set generator adapted by M. Zaki, available in his website (http://www.cs.rpi.edu/~zaki). The data sets were generated to evaluate the performance of the algorithms with respect to the number of samples, and the number of items in the data set and the density of the data set. Like Zhao (2006), we fix standard values for those parameters and for each experiment only the evaluated parameter was changed. Default values of the parameters are shown in Table 7. In Table 7, *minsup* and *maxsup* concern the support constraints considered by the algorithm, while *max_item* concerns the number of elements in each minimal generator. Although TitanicOR does not include this constraint explicitly, BLOSOM does; so we consider it here. We add another test to the loop in line 14, Algorithm 1 to verify if the size of the minimal generator k is lower than or equal to the value of *max_item* only for experimental purposes.

As mentioned before, we consider only the first three parameters in Table 7 when generating each data set. We fix the number of items and the density of the data set, and vary the number of samples between 100, 200, 300, 400, and 500. We then fix the number of samples and the density and vary the number of items between 40, 45, 50, 55, and 60. Finally, we fix the number of samples and the number of items and vary the density between 0.2, 0.3, 0.4, 0.5, and 0.6. For each of those possible configurations, we generate one hundred different data sets in order to reduce bias towards one specific data set with a given configuration; for example, we generate one hundred different data sets with 200 samples, 50 items, and density of 0.5.

Figures 1a, 1b and 1c show the CPU-time measured for both algorithms for the different experiments. Each point corresponds to the average CPU-time measured for each of the one hundred different data sets with the considered configuration; the error bars correspond to the standard deviation. We observe that in all cases TitanicOR has outperformed BLOSOM (although they exhibit almost the same behavior with the different parameters, besides the difference in scales).

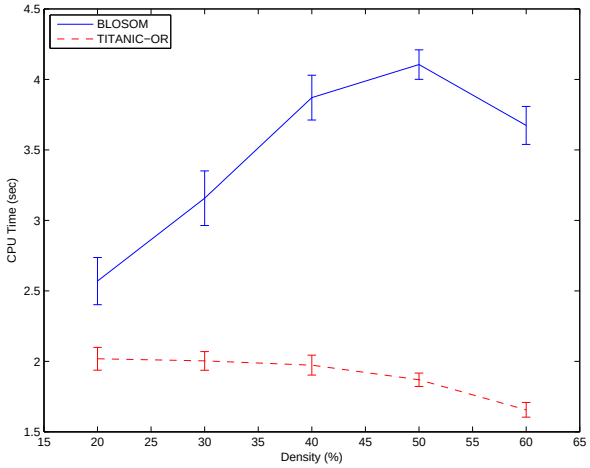
Considering the number of samples (Figure 1a), both algorithms have their time increased while increasing the number of samples, until they reach a peak and the time stabilizes. While the time stabilizes only when the number of samples reaches 400 for BLOSOM, it stabilizes when the number of samples is 200 for TitanicOR. This is unexpected, as Zhao (2006) used the *extraset* data structure to implement BLOSOM and it requires only one database pass to compute the support of minimal generators. On the other hand, the behavior of TitanicOR was completely expected, due to the improvements in the support computation that we suggested. We can also deduce that BLOSOM's performance is more susceptible to variations in the data sets than TitanicOR's since the standard deviations measured for BLOSOM are greater than for TitanicOR.

Regarding the number of items (Figure 1b), TitanicOR is again rather less sensitive than BLOSOM. It can be observed that TitanicOR is roughly two times faster than BLOSOM in all the experiments. The problem resides in the nature of the algorithms. BLOSOM does a depth-first search (DFS) in the disjunctive minimal generators search space, while TitanicOR does a breadth-first search (BFS). Despite several studies pointing out that DFS algorithms are more effective in mining frequent (closed) itemsets, we noticed that BFS algorithms are more suitable for mining



(a) Number of samples

(b) Number of items



(c) Density

Figure 1: Performance measured for the algorithms varying the different parameters

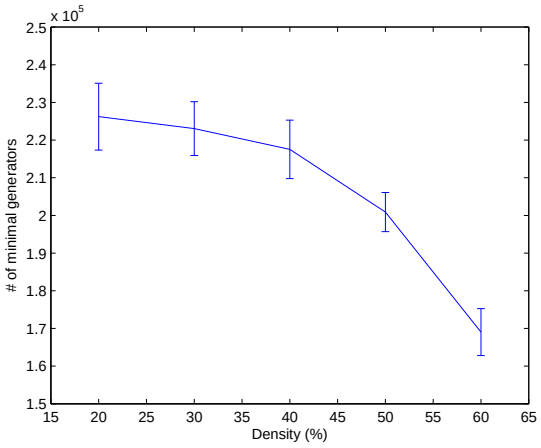


Figure 2: Number of itemsets mined by the algorithms with different densities

minimal generators simply because when traversing the search space breadth-wise, one reaches the smallest sets first. However, the behavior of both algorithms may restrict their application in new domains such as bioinformatics where the number of items are considerably greater than in our tests.

Concerning the density, the performance was as expected. It grows for both until around 50% then starts dropping off. When the density is low, the number of items per sample is also low, thus the samples share few items and so bigger itemsets are required to describe more samples. When the density is high, the probability that an item occurs in a sample is higher. Therefore, smaller itemsets are sufficient to describe the samples. Figure 2 illustrates this situation.

Finally, we investigated the influence of the maximum support, minimum support, and maximum number of items per itemset thresholds in the algorithms using real data sets. We use four real data sets publicly available on the UCI Machine Learning Data Repository (http://archive.ics.uci.edu/ml/ last accessed 2 June 2010); we describe the data sets in Table 8. The number of attributes in Table 8 corresponds to the number of attributes in the data set after binarization since data sets contain categorical attributes.

Table 8: Description of the real data sets used for evaluating the performance of TitanicOR and BLOSOM with different maximum support, minimum support, and maximum number of items per itemset thresholds

Dataset	Number of samples	Number of attributes	density	Att. per sample
Car Evaluation	1728	25	28.0%	7
Chess (King-Rook vs. King-Pawn)	3196	75	49.3%	37
Connect-4	67557	129	33.3%	43
Mushroom	8124	128	18.0%	23

We describe the configuration used to evaluate the influence of each threshold in Table 9; there, “Test 1” corresponds to the configuration used to evaluate the impact of maximum support; “Test 2” corresponds to the configuration used for testing the influence of minimum support; and “Test 3” corresponds to the configuration used for testing the impact of maximum number of items per itemset on the algorithms. The results of each test are respectively shown in Figures 3, 4, and 5.

We present the results of testing the impact of variations of maximum support threshold in the algorithms in Figure 3. The maximum support threshold is used to both reduce the search space and avoid uninteresting itemsets;

Table 9: Description of the configurations used for evaluating the impact of maximum support, minimum support, and maximum number of items per itemset thresholds in TitanicOR and BLOSOM. In this table, $log2(avgtlen)$ denotes the logarithm base two of the number of attributes per sample in a data set. "Test 1" corresponds to the configuration used to evaluate the impact of maximum support; "Test 2" corresponds to the configuration used for testing the influence of minimum support; and "Test 3" corresponds to the configuration used for testing the impact of maximum number of items per itemset on the algorithms

	Max Supp	Min Supp	Max Item
Test 1	10% – 100%	0	4
Test 2	100%	10% – 80%	4
Test 3	90%	20%	$2 - 2^{log2(avgtlen)}$

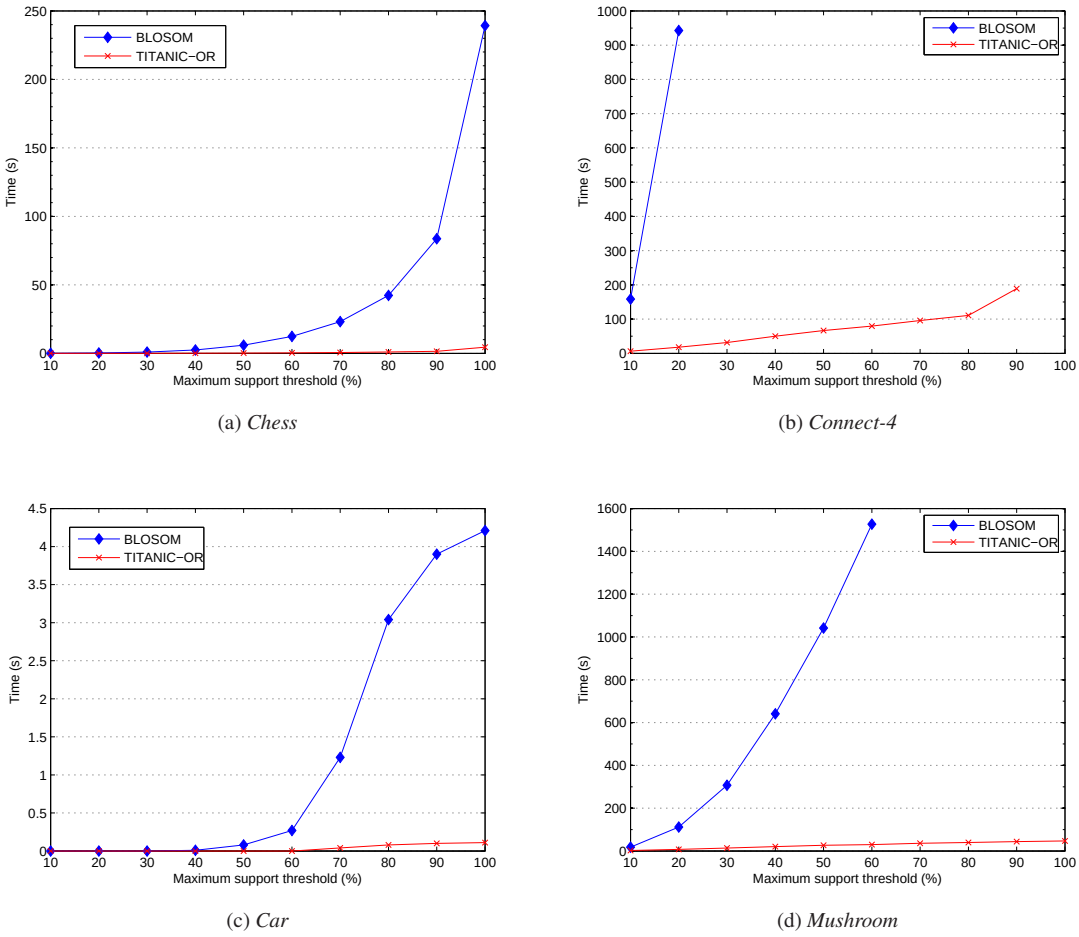
itemsets with high supports may be uninteresting for the user because they cover a large amount of the set of samples, and thus they are likely to be already known. However, if we disregard the assumption that the user may not be interested in high frequent itemsets, then testing the algorithms with variations in maximum support is interesting for evaluating their behavior with different size search spaces. Note that the higher the maximum support, the bigger the search space (as more itemsets satisfy the constraints). We observe that, in general, BLOSOM is much more sensitive to the maximum support threshold than TitanicOR. BLOSOM could not complete the whole test for two data sets, Connect-4 and Mushroom, as shown in Figures 3b and 3d, while TitanicOR could not finish the test with the data set Connect-4 and a maximum support threshold of 100 percent (Figure 3b); both algorithms could not complete their tests due to absence of free memory. We also notice that, in general, TitanicOR is two orders of magnitude less time consuming than BLOSOM; this fact is explained by the time spent by each algorithm on generation of their candidate minimal generators. TitanicOR roughly spends two orders of magnitude less time generating a candidate on average than BLOSOM (Table 10).

Table 10: Average number of itemsets, candidates, and average time spent per candidate by each algorithm for each real data set regarding the variations in maximum support threshold test

		TitanicOR			BLOSOM		
Data set	#Isets	Time(s)	#Cand	Time/Cand	Time(s)	#Cand	Time/Cand
Car	4495.8	0.033	5980.7	2.38E-06	1.27	4497.1	1.62E-04
Chess	84591.5	0.845	115191.8	1.19E-05	41.1	85255.8	3.02E-04
Connect-4	1.99E+06	71.98	2.13E+06	3.26E-05	550.53	3.3E+05	1.44E-03
Mushroom	2.42E+06	26.75	2.77E+06	9.54E-06	607.76	1491512	3.13E-04

Figure 4 contains the results of the test of the influence of minimum support threshold on algorithms. As expected, both algorithms suffer less influence from minimum support than from maximum support; the times spent by the algorithms are much lower considering the variations in minimum support than considering the variations in maximum support. Although minimum support has less impact on the performance of the algorithms, it is still an important feature to reduce the search space and computing time, since we observe a decrease in CPU-time while increasing the minimum support threshold for all data sets in Figure 4.

The maximum number of items per itemset affects the depth that the algorithms can go to when traversing the search space to check candidates; higher thresholds yield more candidates. This test is particularly interesting as it shows the difference in performance between the breadth-first approach used by TitanicOR and the depth-first approach used by BLOSOM. We present the results of the test investigating the influence of the restriction in the number of items per itemset in Figure 5. We notice that both algorithms are sensitive to the increase in the number of items allowed per itemset. As we expected, the depth-first approach used by BLOSOM is much more sensitive to increases in that parameter than the breadth-first approach used by TitanicOR for mining minimal generators. Once



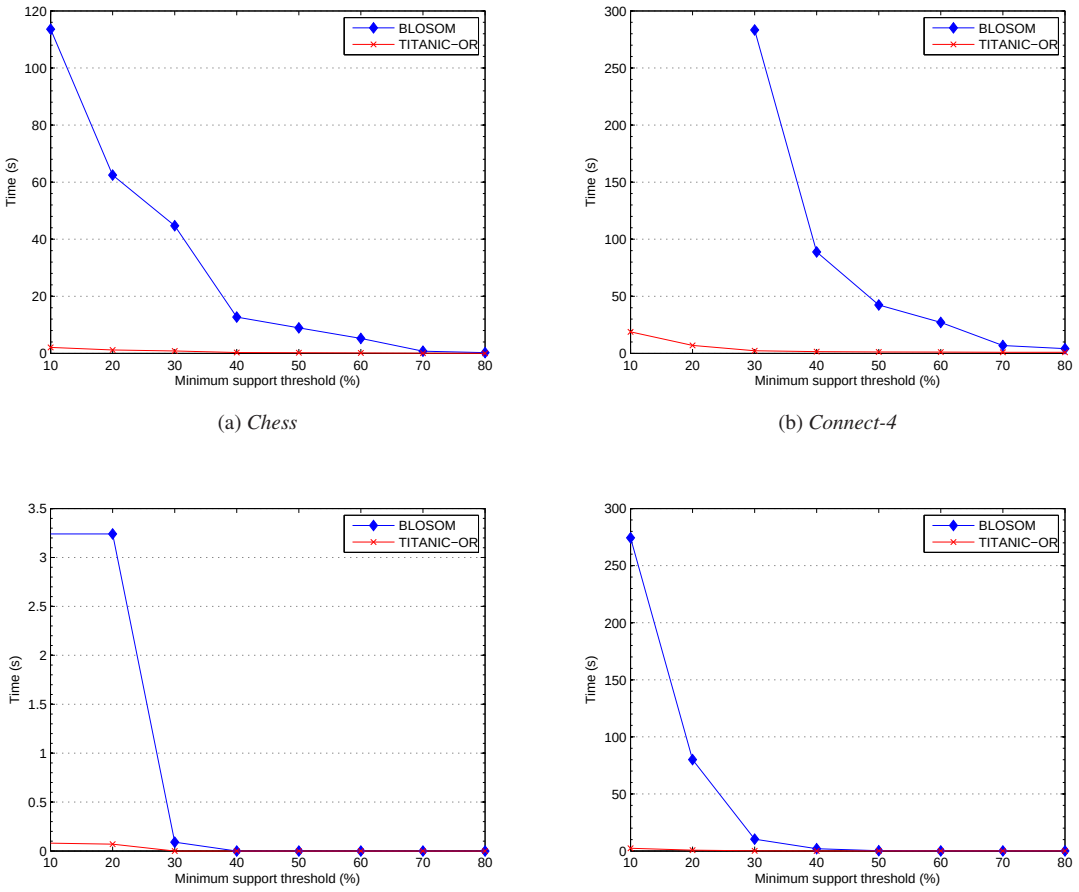
(a) *Chess*

(b) *Connect-4*

(c) *Car*

(d) *Mushroom*

Figure 3: Execution times regarding variations in maximum support for distinct real data sets



(a) *Chess*

(b) *Connect-4*

(c) *Car*

(d) *Mushroom*

Figure 4: Execution times regarding variations in minimum support for distinct real data sets

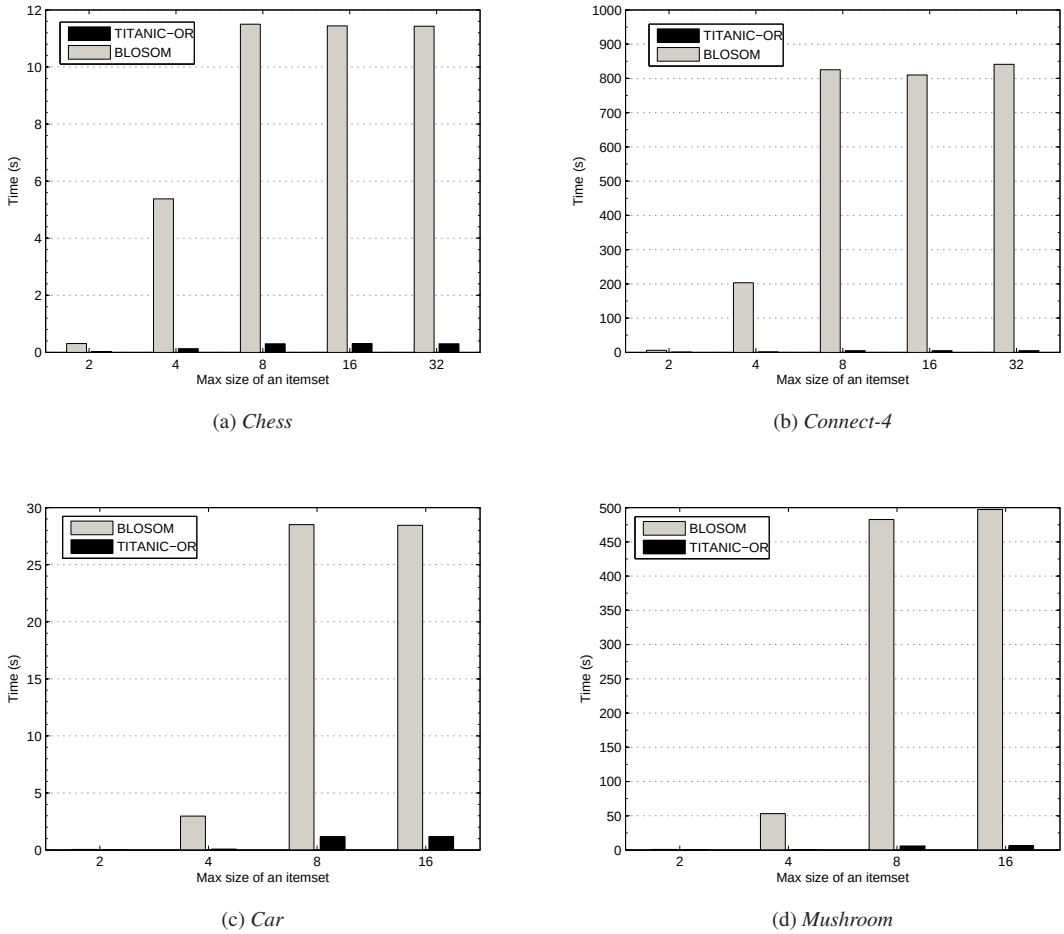


Figure 5: Execution times regarding variations in the maximum number of items per itemset for distinct real data sets

again, TitanicOR spent two orders of magnitude less time per candidate in average than BLOSOM (see Table 11).

5. Related works

5.1. Other methods to mine general association rules

Srikant and Agrawal (1995) proposed the extension of association rules to cover taxonomies over the set of attributes. The problem is called mining generalized association rules and it consists of mining association rules not only considering the attributes by themselves but also considering the categories behind them. For example, considering that sneakers and sandals are types of shoes, and that T-shirts and trousers are types of clothes, it might be more interesting to find rules like "customers that buy shoes tend to buy clothes" than simply "customers that buy sandals tend to buy T-shirts". Rules of the first type are particularly interesting for expanding the expressive power of association rules since they can be interpreted as "customers that buy sneakers or sandals tend to buy trousers or T-shirts". Although this approach apparently extends the interpretation of patterns to cover also disjunctions, it is not the final aim of the authors and, consequently, the disjunctions appearing in rules are only within a single category, *i.e.*, considering attributes from different categories, patterns are still interpreted as conjunctions.

Savasere et al. (1998) introduced the problem of mining interesting *negative association rules*. Negative association rules are rules associating sets of attributes not frequently related, or, in the context that Savasere et al. (1998)

Table 11: Average number of itemsets, candidates, and average time spent per candidate by each algorithm for each real data set regarding the variations in maximum number of items per itemset threshold test

		TitanicOR			BLOSOM		
Data set	#Isets	Time (s)	#Cand	Time/#Cand	Time (s)	#Cand	Time/#Cand
Car	39924	0.6	90861	5.17E-06	14.995	39983.75	3.14E-04
Chess	13561.6	0.214	33834.4	1.17E-05	8.012	13592.4	5.80E-04
Connect-4	38561.8	3.432	91875.8	2.27E-04	537.17	38565.4	1.31E-02
Mushroom	1.7E+05	3.17	3.26E+05	2.16E-05	258.45	1.8E+05	1.34E-03

introduced them, they are rules relating items from a store that are not frequently bought together. Those rules involves a positive antecedent and a negative consequent. Therefore, the problem involves mining both positive patterns (conjunctive frequent patterns) and negative patterns (conjunctive infrequent patterns). Since the number of negative patterns (patterns that do not occur frequently in the data set) may be exponentially high, Savasere et al. (1998) devised an interestingness measure of rules to prune uninteresting patterns. This measure evaluates the level of unexpectedness when two patterns are negatively associated. Although they introduced such negative interpretation, patterns are still treated as conjunctions, *i.e.*, negative patterns are simply conjunctive interpreted patterns that are not frequent in the data set.

Nanavati et al. (2001) discuss the problem of mining generalized disjunctive association rules. They propose the extension of association rules to cover not only conjunctions but also inclusive and exclusive disjunctions. Although they devised an algorithm to find such disjunctive rules, they assume that the set of patterns is given by the domain expert or found by a frequent pattern mining algorithm. Therefore, they provide more expressive power only for association rules; patterns still have the same conjunctive interpretation.

5.2. Other methods to mine disjunctive patterns

The proposed extensions we have seen up to now are concerned with the interpretation of association rules and not the interpretation of the patterns. To the best of our knowledge, the first approach to extend the way the patterns are interpreted was introduced by Zhao et al. (2006). They introduced *BLOSOM*, a mathematical framework to mine general Boolean expressions. Zhao et al. (2006) expanded the definition of closed patterns and minimal generators to cover Boolean expressions containing conjunctions, disjunctions and negations. BLOSOM also includes a CHARM-based (Zaki and Hsiao, 2002) algorithm to mine minimal generators and closed patterns in the disjunctive normal form and in the conjunctive normal form.

Hamrouni et al. (2009) thoroughly investigate the disjunctive interpretation of patterns proposed by Zhao et al. (2006). They improved the definitions of the functions that characterize disjunctive closed patterns and also demonstrated several properties of both the functions and the patterns. One of the most important demonstrated features of disjunctive (closed) patterns is their capacity to concisely represent the set of all conjunctive patterns. This feature reaffirms the necessity of developing new algorithms to efficiently mine this type of patterns.

5.3. Methods and Applications in other domains

Varadan and Anastassiou (2006) proposed a framework to discover Boolean expressions describing diseased tissues. To infer a Boolean expression from a microarray expression data, first, the authors propose the discretization of the data. Rather than discretizing the gene expression values individually, Varadan and Anastassiou (2006) use a single threshold for the whole set of genes. They believe that, while defining thresholds for genes individually is good for single gene analysis, defining a single threshold for the whole set of genes is more convenient for the analysis of the correlations among genes. After the data discretization, the gene subset selection process starts. The authors propose the use of an entropy-based feature selection method. First, they determine the number of genes to be included in the subset. Thereafter, they enumerate all the possible combinations of genes and calculate their entropy. The subset of genes with the lowest entropy is retained. To calculate the entropy, for each subset of genes, the authors enumerate all the possible combination of values for the genes (called states of the gene set) and count the relative frequency

of each state with respect to diseased and healthy samples. Selected the subset of genes the minimizes the entropy, the last stage is the extraction of the Boolean expressions describing the disease. The final Boolean expression is the result of an algorithm that is inputed with the disjunction of the expressions of each state and outputs its equivalent minimal expression. Although this is an interesting approach, as the authors point out several results obtained with it applied to a prostate cancer data set, the whole process is prohibitively expensive since it involves several intrinsically exponential subtasks.

Sahoo et al. (2008) discover Boolean implication networks from large microarray data sets of humans, mice, and fruit flies. Boolean implication networks are graphs built from genes and the binary relation between a pair of genes, *i.e.*, whether the low-high expression of one gene determines or is associated with the low-high expression of another gene. To obtain a Boolean implication for each pair of genes, Sahoo et al. (2008) plot a scatter graph, specify the thresholds to determine which values represent low-high values for each gene, and finally analyze the graph. After they obtain the Boolean implications, they construct the graph. Such networks are interesting since they reveal gene interactions in each species. Although this is a relevant work, it may be the case that not only a single gene influence another, but a set of genes influencing another set. Those relations between sets of genes may not be observed performing the analysis proposed by Sahoo et al. (2008), however, this can be done with association analysis.

Thummalapenta and Xie (2011) propose a framework for detecting frequent patterns in source codes to discover programming rules. Those programming rules are useful to analyze source codes and detect regions that may be susceptible to fail. Their framework, called Alattin, includes an algorithm for mining inclusive and exclusive disjunctive frequent itemsets beyond the traditional conjunctive ones. They showed that the use of disjunctive patterns reduce the number of false positives when detecting regions susceptible to fail. Their algorithm is majorly inspired on the traditional Apriori approach, although they proposed a modification to greedily select candidates with the highest support. Their work differs from ours because they mine all possible frequent disjunctive patterns while we identify only the minimal generators.

Goel et al. (2010) propose the use of generalized Boolean expression as constraints to be used in sequential circuit equivalence checking algorithms. They verified that the use of Boolean expressions in those algorithms can reduce the search space. This fact allowed the authors check large models that could not be checked before the use of those new constraints. The Boolean expressions included in the algorithms were mined with BLOSOM. They considered only minimal generators because these expressions contains the same information regarding the support as the closed expressions but, at the same time, they are the simplest way of representing the latter.

6. Conclusion and Future Works

We revisited the disjunctive closed itemsets proposed by Zhao et al. (2006), and we proved that the disjunctive support function, also proposed by them, is a weight function compatible with the mentioned closure operator according to the definition stated by Stumme et al. (2002). That proof allowed us to adapt the original version of Titanic algorithm (Stumme et al., 2002), and create TitanicOR that uses that weight function to compute disjunctive minimal generators.

We evaluated the performance of TitanicOR using both synthetic and real data sets. We used synthetic data to investigate the algorithm's behavior for data with different numbers of transactions, different numbers of items, and different densities; while we used real data to investigate the influence of minimum support, maximum support, and maximum size of itemsets thresholds on TitanicOR's computing time. In order to evaluate the performance of our algorithm, we compared the CPU-times measured for TitanicOR with these data sets with the CPU-times measured for BLOSOM, the state of the art method and the only one available until TitanicOR for computing disjunctive minimal generators. We observed that TitanicOR's computing time is considerably better than BLOSOM's, with TitanicOR being up to two orders of magnitude faster than the latter.

Regarding the tests with real data, one that deserves our attention is where the maximum size of itemsets is varied. That test is useful for evaluating and enhancing the differences between the breadth-first and depth-first approach used respectively by TitanicOR and BLOSOM. We notice that the breadth-first approach used by our method spends less time per candidate than the depth-first approach; this resulted once again in TitanicOR spending two orders of magnitude less time computing minimal generators than BLOSOM. This strongly suggests that the differences in computing time are not due to implementation issues, but are indeed due to the different approaches. We conclude that breadth-first approach is more suitable for computing minimal generators.

We strongly believe that the task of mining general Boolean expressions can be extremely important to mine useful patterns in the Bioinformatics domain as pointed out in Section 5.3. However, traditionally, the algorithms for frequent pattern mining are formulated for data bases containing many samples and few attributes, while the data bases in Bioinformatics contain many attributes and few samples. As we saw in Section 4, the algorithms for mining Boolean expressions are still very sensitive to the number of attributes turning their application in biological data bases infeasible. Therefore, there is a demand to create new algorithms suitable for this new domain. In future works, we intend to develop new algorithms that explore the sample search space to mine disjunctive itemsets.

References

L. Zhao, M. J. Zaki, N. Ramakrishnan, BLOSOM: a framework for mining arbitrary boolean expressions, in: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, ISBN 1-59593-339-5, 827–832, doi:http://doi.acm.org/10.1145/1150402.1150511, 2006.

G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal, Computing iceberg concept lattices with TITANIC, Data Knowl. Eng. 42 (2) (2002) 189–222, ISSN 0169-023X, doi:http://dx.doi.org/10.1016/S0169-023X(02)00057-5.

R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, ISBN 0-89791-592-5, 207–216, doi: http://doi.acm.org/10.1145/170035.170072, 1993.

D. Sahoo, D. L. Dill, A. J. Gentles, R. Tibshirani, S. K. Plevritis, Boolean implication networks derived from large scale, whole genome microarray datasets, Genome Biology 9 (2008) R157+, ISSN 1465-6906, doi:http://dx.doi.org/10.1186/gb-2008-9-10-r157.

V. Varadan, D. Anastassiou, Inference of disease-related molecular logic from systems-based microarray analysis., PLoS computational biology 2 (6), ISSN 1553-7358, doi:http://dx.doi.org/10.1371/journal.pcbi.0020068.

T. Hamrouni, S. Ben Yahia, E. Mephu Nguifo, Sweeping the disjunctive search space towards mining new exact concise representations of frequent itemsets, Data Knowl. Eng. 68 (10) (2009) 1091–1111, ISSN 0169-023X, doi:http://dx.doi.org/10.1016/j.datak.2009.05.001.

R. Srikant, R. Agrawal, Mining Generalized Association Rules, in: VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-379-4, 407–419, 1995.

A. Savasere, E. Omiecinski, S. B. Navathe, Mining for Strong Negative Associations in a Large Database of Customer Transactions, in: ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, ISBN 0-8186-8289-2, 494–502, 1998.

A. A. Nanavati, K. P. Chitrapura, S. Joshi, R. Krishnapuram, Mining generalised disjunctive association rules, in: CIKM '01: Proceedings of the tenth international conference on Information and knowledge management, ACM, New York, NY, USA, ISBN 1-58113-436-3, 482–489, doi:http://doi.acm.org/10.1145/502585.502666, 2001.

S. Thummalapenta, T. Xie, Alattin: mining alternative patterns for defect detection, Automated Software Engineering (2011) 1–31ISSN 0928-8910, URL `http://dx.doi.org/10.1007/s10515-011-0086-z`, 10.1007/s10515-011-0086-z.

Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal, Mining frequent patterns with counting inference, SIGKDD Explor. Newsl. 2 (2) (2000) 66–75, ISSN 1931-0145, doi:http://doi.acm.org/10.1145/380995.381017.

L. Zhao, Mining subspace and boolean patterns from data, Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY, USA, adviser-Zaki, Mohammed Javeed, 2006.

B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 3540627715, translator-Franzke, C., 1997.

R. Agrawal, R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in: VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-153-8, 487–499, 1994.

M. J. Zaki, C.-J. Hsiao, CHARM: An Efficient Algorithm for Closed Itemset Mining, in: R. L. Grossman, J. Han, V. Kumar, H. Mannila, R. Motwani (Eds.), SDM, SIAM, ISBN 0-89871-517-2, 2002.

N. Goel, M. Hsiao, N. Ramakrishnan, M. Zaki, Mining Complex Boolean Expressions for Sequential Equivalence Checking, in: Test Symposium (ATS), 2010 19th IEEE Asian, ISSN 1081-7735, 442 –447, doi:10.1109/ATS.2010.81, 2010.

* We present a new algorithm for mining disjunctive minimal generators
* Disjunctive minimal generators are new interpretations of itemsets
* Our approach is two orders of magnitude faster than the state of the art
* We conclude that the breadth-first approach is more suitable for minimal generators