DBC: A Condensed Representation of Frequent Patterns for Efficient Mining [⋆,⋆⋆]

Artur Bykowski [a] Christophe Rigotti [a,*]

[a] *Laboratoire d'Ingénierie des Systèmes d'Information, INSA Lyon,*
Bâtiment Blaise Pascal, F-69621 Villeurbanne Cedex, France

Abstract

Given a large set of data, a common data mining problem is to extract the frequent patterns occurring in this set. The idea presented in this paper is to extract a condensed representation of the frequent patterns called disjunction-bordered condensation (DBC), instead of extracting the whole frequent pattern collection. We show that this condensed representation can be used to regenerate all frequent patterns and their exact frequencies. Moreover, this regeneration can be performed without any access to the original data. Practical experiments show that the DBC can be extracted very efficiently even in difficult cases and that this extraction and the regeneration of the frequent patterns is much more efficient than the direct extraction of the frequent patterns themselves. We compared the DBC with another representation of frequent patterns previously investigated in the literature called frequent closed sets. In nearly all experiments we have run, the DBC have been extracted much more efficiently than frequent closed sets. In the other cases, the extraction times are very close.

Key words: data mining, frequent patterns, condensed representations

1 Introduction

An important data mining problem is to extract efficiently the frequent patterns occurring in a large data set. In this paper, we consider the extraction of

[⋆] Recommended by Nikolaos Koudas.

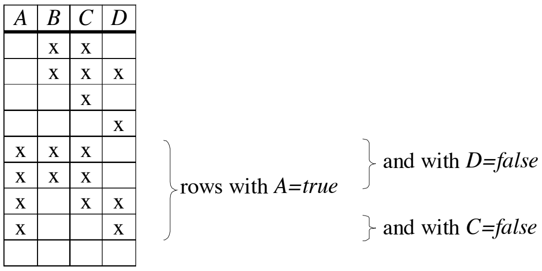
[⋆⋆] This research is part of the cInQ project (IST 2000-26469) that is partially funded by the European Commission IST Programme - Future and Emergent Technologies.

[*] Corresponding author.

Email addresses: Artur.Bykowski@lisi.insa-lyon.fr (Artur Bykowski), Christophe.Rigotti@lisi.insa-lyon.fr (Christophe Rigotti).

A	*B*	*C*	*D*
x	x	x	
	x	x	x
x			x
			x
		x	
x		x	x
	x	x	
x	x	x	
	x		x

Fig. 1. Baskets of customers.



A	*B*	*C*	*D*
	x	x	
	x	x	x
		x	
			x
x	x	x	
x	x	x	
x		x	x
x			x

rows with *A*=*true*
and with *D*=*false*
and with *C*=*false*

Fig. 2. Illustration of simple disjunctive rules.

frequent patterns called *frequent itemsets* which are commonly used to derive efficiently another popular set of patterns called *association rules* [1].

The problem of the frequent itemset extraction can be shortly stated as follows in the context of *basket analysis*, a typical data mining task. A collection of purchases of customers is encoded in a table, where each row represents a customer basket (i.e., a set of items purchased together). A toy example, with four items A, B, C and D, and eight baskets (transactions) is depicted in Figure 1. We use 'x' to denote occurrences of the items in the baskets. Let r be such a table and X be a set of items. The *support* of X in r denoted $Sup(r, X)$ is the number of baskets in r containing all items in X. Then the frequent itemset mining problem is to find all pairs $\langle X, Sup(r, X)\rangle$ such that $Sup(r, X)$ exceeds a given threshold. In the following, every time we say that we extract frequent itemsets, we mean extracting all frequent itemsets along with their supports.

Example 1 *If we consider the baskets represented in Figure 1 and a support threshold value of 2, then the solution to the frequent itemset mining problem is the collection of pairs* $\langle\emptyset, 9\rangle$, $\langle\{A\}, 4\rangle$, $\langle\{A, B\}, 2\rangle$, $\langle\{A, B, C\}, 2\rangle$, $\langle\{A, C\}, 3\rangle$, $\langle\{A, D\}, 2\rangle$, $\langle\{B\}, 5\rangle$, $\langle\{B, C\}, 4\rangle$, $\langle\{B, D\}, 2\rangle$, $\langle\{C\}, 6\rangle$, $\langle\{C, D\}, 2\rangle$ *and* $\langle\{D\}, 5\rangle$.

We address the problem of frequent itemset extraction within data sets composed of a single binary table, i.e. table where the attributes can take 1 of 2 different values: *true* or $false$ (in the basket analysis context, the value states if an item occurred).

The number of rows and the number of items in the binary data sets corresponding to real problems are rather large and naïve solutions can not be used. Several different techniques have been proposed to perform this task on large data sets (e.g., [2–6]). Dealing with data sets that are made up of multiple tables has also been investigated, e.g., in [7,8]. The extraction of frequent itemsets within data sets containing attributes other than binary (e.g. real) may be accomplished by first *binarizing* the data — as pointed by Liu et al. in [9], good techniques for this purpose may be found in [10,11].

An alternative promising approach has been developed in [12–14], based on the following idea: instead of mining all frequent patterns, it is sufficient to extract a particular subset of the frequent pattern collection, such that we can regenerate from this subset the whole collection. In this paper, we call such a subset a *condensed representation*[1] of the frequent pattern collection.

Ideally, a condensed representation is much smaller than the original collection and can be extracted more efficiently, while allowing a quick regeneration of all frequent patterns without costly scan of the original data and new support counting.

To our knowledge, two condensed representations have been proposed in the literature for frequent itemsets: *closed sets* [12] and *δ-free sets* [13,14]. The first framework allows regenerating exactly the collection of frequent patterns and their frequencies. The second offers two possibilities: an exact or an approximate regeneration of the frequencies, corresponding respectively to $\delta = 0$ and $\delta > 0$. In the exact case, closed sets and δ-free sets (with $\delta = 0$) present similar benefits. δ-free sets with $\delta > 0$ lead to a much smaller representation and to a more efficient extraction, but at the cost of some uncertainty on the frequencies of regenerated itemsets.

In this paper, we consider the exact regeneration of frequencies and we propose a new condensed representation called *disjunction-bordered condensation* (DBC). This representation contains the same amount of information as the full collection of frequent itemsets, but consists of a subcollection of the frequent itemsets and can be extracted more efficiently. The core of the contribution of this paper is the DBC representation and the elementary procedures to extract it and to derive frequent itemsets and frequent closed sets. Several experiments are presented and show the practical interest of the DBC to find frequent patterns, especially in difficult cases. The experiments show that the DBC is an interesting condensed representation of frequent itemsets but also of frequent closed sets themselves. Thus DBC can be used to greatly reduce the running time and the storage space requirement of the data mining processes involving frequent itemsets and frequent closed sets. This immediately enables useful applications of the DBC like more efficient association rule [1] extraction or faster computation of rule summaries (e.g., [16]).

This paper is a major extension of [17], where preliminary results have been presented.

Informal description of the DBC.

The condensation of the collection of frequent itemsets by the DBC is due to a

[1] We borrow this term from [15].

property that binds supports of some itemsets by equations. This property is based on expressions called simple disjunctive rules. The general form of such rules is $A_1 \wedge A_2 \wedge \ldots \wedge A_{n-2} \Rightarrow A_{n-1} \vee A_n$, where A_i represent different items (with an exception on the last 2 items, i.e. A_{n-1} and A_n, which are allowed to be the same). This rule states that if $A_1, \ldots, A_{n-2}$ are set to $true$ within a row, then A_{n-1} or A_n is set to $true$ within the same row. The rule may hold in a row or not. The latter case arises when A_{n-1} and A_n are set to $false$ in spite of all $A_1, A_2, \ldots, A_{n-2}$ being set to $true$. If one of the elements in the premise (i.e., the left hand side) of the rule is not set to $true$ then (as for implications in propositional logic) the rule is also said to *hold* (since it is not violated).

Consider the table r' depicted in Figure 2. The rule $B \wedge C \Rightarrow A \vee D$ holds for example in the second and third row, but not in the first one. The DBC is based on rules that hold in all rows of a table, as for instance $A \Rightarrow C \vee D$ in r'.

Observe the rows with A $true$ in Figure 2. Since $A \Rightarrow C \vee D$ holds in all rows, there is no row with A $true$ and both C and D $false$. Thus, the support of A is simply equal to the sum of the supports of $\{A, C\}$ and $\{A, D\}$ minus the support of $\{A, C, D\}$ (because the support of $\{A, C, D\}$ has been counted in the support of both $\{A, C\}$ and $\{A, D\}$). So if we know the supports of $\{A\}$, $\{A, C\}$ and $\{A, D\}$, then we can avoid the extraction and the support counting of $\{A, C, D\}$ by simply using the equation $Sup(r', \{A, C, D\}) = Sup(r', \{A, C\}) + Sup(r', \{A, D\}) - Sup(r', \{A\})$.

Moreover, if $A \Rightarrow C \vee D$ holds in all rows, then $\alpha \Rightarrow C \vee D$ holds also in all rows for any conjunction α containing A. Such a rule is for example, in Figure 2, $A \wedge B \Rightarrow C \vee D$, and we have $Sup(r', \{A, B, C, D\}) = Sup(r', \{A, B, C\}) + Sup(r', \{A, B, D\}) - Sup(r', \{A, B\})$. This property is very interesting since for the dataset of Figure 2 we can then skip the support counting of all proper supersets of $\{A, C, D\}$, while still being able to compute their supports using these relations between the itemset supports.

An itemset containing items that can be used to form a simple disjunctive rule holding in all rows is called a *non-disjunction-free set*. The remaining itemsets are called *disjunction-free sets*. In the previous example $\{A, C, D\}$ and $\{A, B, C, D\}$ are non-disjunction-free, because of the rule $A \Rightarrow C \vee D$.

A DBC is simply the collection of all frequent disjunction-free sets and of all frequent and minimal (w.r.t. set inclusion) non-disjunction-free sets, along with the corresponding support information.

The most interesting property of the DBC is that this collection of itemsets can be extracted very efficiently and is sufficient to compute all frequent itemsets and their supports.

Organization of the paper.

The next section contains preliminary definitions used in this paper. In Section 3, we define the notion of disjunction-free set, and show that these sets can be used as a condensed representation for frequent itemsets. Section 4 describes an algorithm regenerating all frequent itemsets in an efficient manner. The following section reports the size gain of the DBC over frequent itemsets and frequent closed sets. In Section 6, we present both breadth-first and depth-first algorithms to extract the DBC and we describe practical experiments showing that the DBC can be extracted efficiently and, in fact, more efficiently than frequent closed sets. In Section 7, we will show that, in most cases, it is more efficient to mine the DBC and to convert it to frequent closed sets than to directly mine frequent closed sets themselves. Finally, we conclude with a summary.

2 Preliminary definitions

When possible, we follow the notational conventions and definitions of [15,18]. In particular, we use multisets to represent collections of rows and given such a multiset r, we write $t \in r$ to denote that a particular row t belongs to r.

Definition 1 ***(binary database)*** Let R be a set of symbols called items. A *row* is a subset of R. A *binary database* r *over* R is a multiset of rows. Additionally, we suppose that a linear order denoted $\prec$ is given over the set of items R.

Example 2 *Figures 1 and 2 present two binary databases over* $\{A, B, C, D\}$*. The linear order* $\prec$ *can be simply* $A \prec B \prec C \prec D$.

Definition 2 ***(support and frequent itemsets)*** We note $\mathcal{M}(r, X) = \{t \in r | X \subseteq t\}$ the multiset of rows matched by the itemset X and $Sup(r, X) = |\mathcal{M}(r, X)|$ the *support* of X in r, i.e., the number of rows matched by X. Let σ be a support threshold (σ is an absolute number of rows), $Freq(r, \sigma) = \{X | X \subseteq R \text{ and } Sup(r, X) \geq \sigma\}$ is the set of all σ*-frequent* itemsets in r.

The following lemma is a fundamental property for the frequent itemset extraction. It states that the functions $\mathcal{M}$ and Sup are *anti-monotone* w.r.t. itemset inclusion. The anti-monotonicity of Sup has been demonstrated in [2] as an efficient pruning criterion.

Lemma 1 *Let* r *be a binary database over* R*, and* X, Y *be sets of items such that* $Y \subseteq X \subseteq R$*. Then* $\mathcal{M}(r, X) \subseteq \mathcal{M}(r, Y)$ *and* $Sup(r, X) \leq Sup(r, Y)$.

Proof. Let t be any row in r that belongs to $\mathcal{M}(r, X)$. From Definition 2, a row t belongs to $\mathcal{M}(r, X)$ if and only if $X \subseteq t$. $Y \subseteq X \subseteq t$ implies $t \in \mathcal{M}(r, Y)$, and thus $\mathcal{M}(r, X) \subseteq \mathcal{M}(r, Y)$. Consequently, $Sup(r, X) \leq Sup(r, Y)$. □

Corollary 1 *Let X, Y be the same as in the lemma. If X is σ-frequent in r, Y is σ-frequent in r. If Y is not σ-frequent, so neither is X.*

In this paper, to keep the presentation concise, we consider the following notational conventions to handle a *generalized* form of itemsets.

Definition 3 ***(generalized itemsets)*** Let R be a set of symbols. The symbols in R will be called *positive* items, and for each positive item $A \in R$ we consider a *negative* item noted $\overline{A}$. $Gen(R) = R \cup \{\overline{A} | A \in R\}$ is the set of generalized items based on R and a subset X of $Gen(R)$ is called a *generalized itemset* based on R. We denote the items appearing positively and negatively as follows, $Pos(X) = \{A \in R | A \in X\}$ and $Neg(X) = \{A \in R | \overline{A} \in X\}$.

We generalize in this context $\mathcal{M}$ and Sup.

Definition 4 ***(generalized support)*** Let r be a binary database over R and X be a generalized itemset based on R. Then $Gen\mathcal{M}(r, X) = \{t \in r | Pos(X) \subseteq t \wedge Neg(X) \cap t = \emptyset\}$ is the multiset of rows matched by X and the support of X in r is $GenSup(r, X) = |Gen\mathcal{M}(r, X)|$.

Intuitively, the rows matched by a generalized itemset X are the rows that contain all positive items in X but none of the items appearing under a negative form in X.

Example 3 *In Figure 1 the generalized support of the three generalized itemsets $\{\overline{A}, B, C\}$, $\{\overline{A}, \overline{B}\}$ and $\{A, B\}$ is 2.*

The matching and support functions are anti-monotone w.r.t. the generalized set inclusion. This is stated by the following lemma.

Lemma 2 *Let r be a binary database over R and X, Y be generalized itemsets based on R. If $Y \subseteq X$ then $Gen\mathcal{M}(r, X) \subseteq Gen\mathcal{M}(r, Y)$ and $GenSup(r, X) \leq GenSup(r, Y)$.*

Proof. Let t be any row in $Gen\mathcal{M}(r, X)$. $t \in Gen\mathcal{M}(r, X)$ implies $Pos(X) \subseteq t$ and $Neg(X) \cap t = \emptyset$. Since $Y \subseteq X$ then $Pos(Y) \subseteq Pos(X)$ and $Neg(Y) \subseteq Neg(X)$, and then we have $Pos(Y) \subseteq t$ and $Neg(Y) \cap t = \emptyset$. Thus, $t \in Gen\mathcal{M}(r, Y)$. The second conclusion is immediate. □

3 Disjunction-free sets

The notion of disjunction-free set is based on a kind of dependency between the items called *simple disjunctive rules* and defined as follows.

Definition 5 ***(simple disjunctive rule)*** Let X be a set of (positive) items, a *simple disjunctive rule* based on X is an expression of the form $Y \Rightarrow A \vee B$, where $Y \subset X$ and $A, B \in X \setminus Y$. Notice that A and B are single items and also that a rule of the form $Y \Rightarrow A \vee A$ is a particular case of simple disjunctive rule. Let r be a binary database over R, where $X \subseteq R$. The simple disjunctive rule $Y \Rightarrow A \vee B$ is *valid* in r if and only if $\mathcal{M}(r, Y) = \{t \in r | t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$.

Example 4 *The following simple disjunctive rules are valid in the binary database depicted in Figure 1:* $\{A\} \Rightarrow C \vee D$*,* $\emptyset \Rightarrow C \vee D$ *and also* $\{A, B\} \Rightarrow C \vee C$*. In this binary database* $\{D\} \Rightarrow A \vee C$ *is not valid.*

Now, we state three lemmas. They are fundamentals needed in the rest of the paper.

Lemma 3 *Let* r *be a binary database over* R*. Let* A *and* B *be single items in* R*, and* $\overline{A}, \overline{B}$ *be their corresponding negative items. Let* Y *be a set of items such that* $Y \subseteq R \setminus \{A, B\}$*. Then the following three propositions are equivalent:*
(1) $Y \Rightarrow A \vee B$ *is a valid simple disjunctive rule in* r*,*
(2) $Gen\mathcal{M}(r, Y \cup \{\overline{A}, \overline{B}\}) = \emptyset$*,*
(3) $GenSup(r, Y \cup \{\overline{A}, \overline{B}\}) = 0$*.*

Proof. From Definition 4 *(2)* is trivially equivalent to *(3)*. Now, we show that *(3)* implies *(1)* and later that *(1)* implies *(2)*.

Suppose $GenSup(r, Y \cup \{\overline{A}, \overline{B}\}) = 0$. By Definition 4, there is no $t \in r$ such that $Y \subseteq t$ and $\{A, B\} \cap t = \emptyset$. Thus, $\forall t \in r, Y \subseteq t \Rightarrow A \in t \vee B \in t$. So, $\mathcal{M}(r, Y) = \{t \in r | t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$ and finally, by Definition 5, $Y \Rightarrow A \vee B$ is valid.

Suppose now that $Y \Rightarrow A \vee B$ is valid. By Definition 5, $\mathcal{M}(r, Y) = \{t \in r | t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$. So, there is no row $t \in r$ such that $Y \subseteq t$ and $A \notin t$ and $B \notin t$. Thus, the multiset $\{t \in r | Y \subseteq t \wedge \{A, B\} \cap t = \emptyset\} = \{t \in r | Pos(Y \cup \{\overline{A}, \overline{B}\}) \subseteq t \wedge Neg(Y \cup \{\overline{A}, \overline{B}\}) \cap t = \emptyset\}$ is empty. And then, by Definition 4, $Gen\mathcal{M}(r, Y \cup \{\overline{A}, \overline{B}\}) = \emptyset$. □

Lemma 4 *Let* r *be a binary database over* R*, and* X, Y *be itemsets such that* $Y \subseteq X \subseteq R$*. Let* $A, B \in Y$*. If* $Y \setminus \{A, B\} \Rightarrow A \vee B$ *is valid in* r *then* $X \setminus \{A, B\} \Rightarrow A \vee B$ *is also valid in* r*.*

Proof. Suppose that $Y \setminus \{A, B\} \Rightarrow A \vee B$ is valid in r. By Lemma 3, $Gen\mathcal{M}(r, (Y \setminus \{A, B\}) \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Then, since X is a superset of Y from Definition 4 and Lemma 2 we deduce that $Gen\mathcal{M}(r, (X \setminus \{A, B\}) \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Whence, using again Lemma 3, we know that $X \setminus \{A, B\} \Rightarrow A \vee B$ is valid. □

Lemma 5 *Let r be a binary database over R, $X \subseteq R$ be an itemset, and A, B be items in X. Then there exists $Y \subset X$ such that $Y \Rightarrow A \vee B$ is a valid simple disjunctive rule based on X, if and only if $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$.*

Proof. Consider $Y \Rightarrow A \vee B$ the valid simple disjunctive rule based on X and $Z = X \setminus \{A, B\}$. By Definition 5, $Y \subseteq Z$ and then from Lemma 4, we deduce that $Z \Rightarrow A \vee B$ is valid.

We can partition the multiset of rows matched by Z into three multisets $Gen\mathcal{M}(r, Z \cup \{A\})$, $Gen\mathcal{M}(r, Z \cup \{\overline{A}, \overline{B}\})$ and $Gen\mathcal{M}(r, Z \cup \{\overline{A}, B\})$. As $Z \Rightarrow A \vee B$ is valid, using Lemma 3 we have $Gen\mathcal{M}(r, Z \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Thus, $Sup(r, Z) = GenSup(r, Z \cup \{A\}) + GenSup(r, Z \cup \{\overline{A}, B\})$. Because $Gen\mathcal{M}(r, Z \cup \{\overline{A}, B\}) = \{t \in r | t \in \mathcal{M}(r, Z \cup \{B\}) \wedge t \notin \mathcal{M}(r, Z \cup \{A, B\})\}$ and $\mathcal{M}(r, Z \cup \{B\}) \supseteq \mathcal{M}(r, Z \cup \{A, B\})$, we have $GenSup(r, Z \cup \{\overline{A}, B\}) = Sup(r, Z \cup \{B\}) - Sup(r, Z \cup \{A, B\})$, and finally $Sup(r, X \setminus \{A, B\}) = Sup(r, X \setminus \{B\}) + Sup(r, X \setminus \{A\}) - Sup(r, X)$.

Suppose now that $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$. Let $Z = X \setminus \{A, B\}$. $GenSup(r, Z \cup \{\overline{A}, \overline{B}\}) = GenSup(r, Z \cup \{\overline{A}\}) - GenSup(r, Z \cup \{\overline{A}, B\}) = (Sup(r, Z) - Sup(r, Z \cup \{A\})) - (Sup(r, Z \cup \{B\}) - Sup(r, Z \cup \{A, B\})) = Sup(r, X \setminus \{A, B\}) - Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A\}) + Sup(r, X) = 0$. Therefore, from the Lemma 3, we deduce that $Z \Rightarrow A \vee B$ is valid, which proves the existence of the required rule. □

Now, we define the notion of disjunction-free sets.

Definition 6 ***(disjunction-free set)*** Let r be a binary database over R, $X \subseteq R$ is a *disjunction-free set* w.r.t. r if and only if no simple disjunctive rule based on X is valid in r. The set of all disjunction-free sets w.r.t. r is noted $DFree(r)$.

Example 5 *In the binary database of Figure 2 $\{A, C, D\}$ is not a disjunction-free set because the rule $\{A\} \Rightarrow C \vee D$ is valid, but $\{A, B\}$ is disjunction-free since there is no valid simple disjunctive rule based on $\{A, B\}$.*

Anti-monotonicity of a property of itemsets w.r.t. itemset inclusion is helpful for an efficient pruning, as the anti-monotonicity of supports (Lemma 1) was in the APRIORI algorithm [2].

The anti-monotonicity of disjunction-freeness follows directly from the definition of disjunction-free sets and is stated by the following lemma.

Lemma 6 *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in DFree(r)$ then $Y \in DFree(r)$.*

Definition 7 ***(frequent disjunction-free set)*** Let r be a binary database over a set of items R, $FreqDFree(r, \sigma) = Freq(r, \sigma) \cap DFree(r)$ denotes the set of all σ-frequent disjunction-free sets w.r.t. r.

We reuse the concept of negative border introduced in [18] and define it for σ-frequent disjunction-free sets. Informally, the negative border consists of the smallest itemsets (w.r.t. set inclusion) that are not σ-frequent disjunction-free sets.

Definition 8 ***(negative border)*** Let r be a binary database over a set of items R, the negative border of $FreqDFree(r, \sigma)$ is noted $\mathcal{B}d^-(r, \sigma)$ and is defined as follows: $\mathcal{B}d^-(r, \sigma) = \{X | X \subseteq R, X \notin FreqDFree(r, \sigma) \wedge (\forall Y \subset X, Y \in FreqDFree(r, \sigma))\}$.

Example 6 *Let us consider again the binary database depicted in Figure 1. Using the support threshold of 1 (row), the collection of all frequent disjunction-free sets is $\mathcal{S} = \{\emptyset, \{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\}$. The itemsets $\{A, B, C\}$, $\{A, C, D\}$, $\{B, C, D\}$ and $\{C, D\}$ are also frequent, but not disjunction-free since the rules $\{A, B\} \Rightarrow C \vee C$ and $\emptyset \Rightarrow C \vee D$ are valid. $\{A, B, D\}$ is disjunction-free but is not frequent. The negative border of $\mathcal{S}$ (i.e., the smallest itemsets not in $\mathcal{S}$) is the collection $\{\{A, B, C\}, \{A, B, D\}, \{C, D\}\}$.*

Out of all itemsets of the negative border, we need only the frequent ones. The definition follows.

Definition 9 ***(frequent itemsets of negative border)*** Let r be a binary database over a set of items R, the collection of all frequent itemsets of negative border of $FreqDFree(r, \sigma)$ is defined as $Freq\mathcal{B}d^-(r, \sigma) = \mathcal{B}d^-(r, \sigma) \cap Freq(r, \sigma)$.

Definition 10 ***(disjunction-bordered condensation)*** *The disjunction-bordered condensation is the collection of frequent itemsets that are either disjunction-free sets or in the negative border. Formally, $DBC(r, \sigma) = \langle FreqDFree(r, \sigma), Freq\mathcal{B}d^-(r, \sigma)\rangle$.*

Example 7 *In the case of the binary database considered in Example 6, the frequent itemsets of the negative border are $\{A, B, C\}$ and $\{C, D\}$. Then the disjunction-bordered condensation for $\sigma = 1$ is $\langle \mathcal{S}, \{\{A, B, C\}, \{C, D\}\}\rangle$.*

We can now state the *correctness* of the DBC, i.e. the sets of the DBC along with their supports are sufficient to determine the support of any frequent itemset.

Theorem 1 *Let r be a binary database over a set of items R, X be an itemset such that $X \subseteq R$, and σ be an absolute support threshold. Using the itemsets in $FreqDFree(r, \sigma)$ and in $Freq\mathcal{B}d^-(r, \sigma)$, together with their supports, we can determine if X is σ-frequent, and when X is σ-frequent we can also determine $Sup(r, X)$.*

Proof. Let X be any itemset. If X is in $DBC(r, \sigma)$ (in $FreqDFree(r, \sigma)$ or in $Freq\mathcal{B}d^-(r, \sigma)$), we know its support, so we can state that X is σ-frequent, and we can also give $Sup(r, X)$. This situation will be referred to as *trivial case* in this proof.

The proof is made by induction on $|X|$.

First, let us consider the case where $X = \emptyset$. If $X \in FreqDFree(r, \sigma)$ then we have the *trivial case*. Otherwise, by Definition 8, $X \in \mathcal{B}d^-(r, \sigma)$. Given that there cannot be simple disjunctive rule based on $\emptyset$ (see Definition 5), X must be disjunction-free and thus is not σ-frequent.

In the following, we suppose that $X \neq \emptyset$.

Hypothesis. Suppose that for every itemset $W \subset X$, we can determine if W is σ-frequent, and when W is σ-frequent we can also determine $Sup(r, W)$.

If $X \in FreqDFree(r, \sigma)$ then we have again the *trivial case.*

If $X \notin FreqDFree(r, \sigma)$ and $\forall Y \subseteq X, Y \notin Freq\mathcal{B}d^-(r, \sigma)$ then we show that X is not σ-frequent. First, observe that if $X \notin FreqDFree(r, \sigma)$ then, by Definition 8, $\exists W \subseteq X$ such that $W \in \mathcal{B}d^-(r, \sigma)$. Let W be such a set. W is not σ-frequent since $\forall Y \subseteq X, Y \notin Freq\mathcal{B}d^-(r, \sigma)$. By the anti-monotonicity of support (Lemma 1), we know that X is not σ-frequent, because $W \subseteq X$.

If $X \notin FreqDFree(r, \sigma)$ and $X \in Freq\mathcal{B}d^-(r, \sigma)$ then, we have once more the *trivial case.*

We consider now the case where $X \notin FreqDFree(r, \sigma)$ and $\exists Y \subset X, Y \in Freq\mathcal{B}d^-(r, \sigma)$. Let Y be such an itemset.

$Y \in Freq\mathcal{B}d^-(r, \sigma)$ implies that Y is not a disjunction-free set. In this case, we now construct a valid simple disjunctive rule based on Y, and then we use this rule to decide if X is σ-frequent and to compute its support. By Definition 6, Y is not a disjunction-free set implies that there exists $Z \subset Y$ and $A, B \in Y \setminus Z$

such that $Z \Rightarrow A \vee B$ is a valid simple disjunctive rule in r. By Lemma 5, we have $Sup(r, Z \cup \{A, B\}) = Sup(r, Z \cup \{B\}) + Sup(r, Z \cup \{A\}) - Sup(r, Z)$.

Since Y is σ-frequent, by Lemma 1 all its subsets are σ-frequent. Because $|Y| < |X|$ and Y is σ-frequent by the induction hypothesis we can determine the supports of all subsets of Y. So we can find among all subsets of Y four itemsets $Z, Z \cup \{A, B\}, Z \cup \{B\}, Z \cup \{A\}$ satisfying the relation $Sup(r, Z \cup \{A, B\}) = Sup(r, Z \cup \{B\}) + Sup(r, Z \cup \{A\}) - Sup(r, Z)$, which corresponds to a valid rule $Z \Rightarrow A \vee B$.

By the induction hypothesis, we can determine for the itemsets $X \setminus \{A, B\}, X \setminus \{A\}$ and $X \setminus \{B\}$ (the items A and B are the same as above) if they are all σ-frequent and if so we can also determine their supports. If these three sets are σ-frequent, using their supports we can determine the support of X as $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$ (according to Lemma 5), because $Z \Rightarrow A \vee B$ is a valid disjunctive rule based on X. If at least one of these itemsets is not σ-frequent so neither is X (by Lemma 1), which completes the proof. □

It should be noticed that the proof of Theorem 1 is constructive and that it can be used as a naïve recursive algorithm to determine $Sup(r, X)$.

The next theorem supports an earlier claim, which stated that the disjunction-bordered condensation is a subcollection of the collection of all frequent itemsets. The proof is trivial, following the definitions of $FreqDFree(r, \sigma)$ and of $Freq\mathcal{B}d^-(r, \sigma)$.

Theorem 2 *$FreqDFree(r, \sigma) \cup Freq\mathcal{B}d^-(r, \sigma) \subseteq Freq(r, \sigma)$.*

4 Regeneration of all frequent itemsets

In this section, we present a levelwise algorithm, called RegenFreq, that regenerates all frequent itemsets from the DBC. Here, we assume that we regenerate $Freq(r, \sigma)$ and the corresponding supports given $DBC(r, \sigma)$ and the corresponding supports. Nonetheless, the frequency thresholds may be different — we might regenerate $Freq(r, \sigma')$, where $\sigma' \geq \sigma$, by skipping from the $DBC(r, \sigma)$ the itemsets whose supports are below σ' and applying the same algorithm to the reduced input.

We give RegenFreq in a detailed form. The actual implementation integrates few straightforward optimizations. Most important hints will be mentioned throughout this section.

The data structure corresponding to one frequent itemset will be called a node. Here is its structure:

itemset	: set of items
support	: integer
prunedby	: set of items

Let $\mathcal{F}req_i$ denote the collection of nodes corresponding to itemsets of size i. A collection of such nodes is stored in an itemset-prefix-tree, a structure allowing efficient access to the node given the value of *itemset* (see [1]).

For a node N, the access to these three fields is denoted respectively N*.itemset*, N*.support* and N*.prunedby*. N*.itemset* corresponds to a frequent itemset[2] and N*.support* is its support.

N*.prunedby* is a set of at most two items. Its meaning if it contains two different items A, B is that $A, B \in N.itemset$ and $N.itemset \setminus \{A, B\} \Rightarrow A \vee B$ is valid. When N*.prunedby* contains only one item A, it means that $A \in N.prunedby$ and $N.itemset \setminus \{A\} \Rightarrow A \vee A$ is valid. Finally, when N*.prunedby* is empty, it corresponds to the case where N*.itemset* is a disjunction-free set. If there are more than one valid rule based on N*.itemset*, N*.prunedby* is supposed to report any of them.

The following algorithm, called FindDisjRule, computes the value of N*.prunedby* for an itemset and will be used in the regeneration of all frequent itemsets.

Algorithm 1 (FindDisjRule)

Input: *Itemset Z, S support of Z, collection of nodes $\mathcal{C}$ including nodes corresponding to all proper subsets of Z of size $|Z| - 1$ and of size $|Z| - 2$.*

Output: *Itemset P corresponding to* prunedby *for Z (see description in text).*

```
Find any $A, B$ items in $Z$ and $N_A, N_B, N_{AB}$ nodes in $\mathcal{C}$ such that
    $N_A.itemset = Z \setminus \{A\}$ and $N_B.itemset = Z \setminus \{B\}$ and $N_{AB}.itemset = Z \setminus \{A, B\}$
    and $N_{AB}.support + S = N_A.support + N_B.support$;
if such $A, B$ exist then // in this case $Z \setminus \{A, B\} \Rightarrow A \vee B$ is valid
    let $P := \{A, B\}$;
else
    let $P := \emptyset$;
fi
output $P$;
```

Now, we recall the algorithm APRIORIGen from [2], which also will be

[2] The field *itemset* is represented in the algorithm for clarity, but using an *itemset*-based prefix-tree structure provides a much more compact storage of the items in this itemset.

used in the regeneration of all frequent itemsets. APRIORIGen realizes the generation of potential frequent itemsets of size $k+1$ given the collection $\mathcal{F}req_k$ of nodes corresponding to all frequent itemsets of size k.

We make use of a notation for accessing individually the items in the frequent itemsets as follows. Let $orderedList(N.itemset)$ be the list of all items in $N.itemset$ sorted in ascending order according to the linear order $\prec$ for items. Then, $N.itemset[i]$ denotes the i^{th} element in the list $orderedList(N.itemset)$.

Algorithm 2 (APRIORIGen)

Input: $\mathcal{F}req_k$ *collection of nodes corresponding to all frequent itemsets of size* k*, with* $k \geq 1$.

Output: $\mathcal{C}_{k+1}$ *collection of itemsets of size* $k+1$ *such that all their subsets of size* k *are represented in* $\mathcal{F}req_k$.

```
insert into $\mathcal{C}_{k+1}$
select $N.itemset[1], N.itemset[2], \ldots, N.itemset[k], R.itemset[k]$
from $\mathcal{F}req_k$ $N$, $\mathcal{F}req_k$ $R$
where $N.itemset[1] = R.itemset[1], N.itemset[2] = R.itemset[2], \ldots,$
        $N.itemset[k-1] = R.itemset[k-1], N.itemset[k] \prec R.itemset[k]$;

for all $C \in \mathcal{C}_{k+1}$ do
    for all $A \in C$ do
        if not $\exists N \in \mathcal{F}req_k$ such that $N.itemset = C \setminus \{A\}$ then
            delete $C$ from $\mathcal{C}_{k+1}$;
        fi
    od
od

output $\mathcal{C}_{k+1}$
```

Both algorithms, FindDisjRule and APRIORIGen will also be used in other algorithms presented in this paper. Now, we give the algorithm RegenFreq itself, which regenerates all frequent itemsets and their supports.

Algorithm 3 (RegenFreq)

Input: σ*,* $DBC(r, \sigma)$ *of the form* $\langle \mathcal{F}, \mathcal{B} \rangle$ *and for each itemset* $X \in \mathcal{F} \cup \mathcal{B}$ *its support* $Sup(r, X)$.

Output: $Freq(r, \sigma)$ *and their supports.*

```
for all $X \in \mathcal{F}$ do // Disjunction-free input sets
    Create new node $N_X$ in $\mathcal{F}req_{|X|}$ with $N_X.support := Sup(r, X)$, $N_X.itemset := X$,
      $N_X.prunedby := \emptyset$;
od
for all $X \in \mathcal{B}$ do // Non-disjunction-free input sets
```

```
5.      Create new node $N_X$ in $\mathcal{F}req_{|X|}$ with $N_X.support := Sup(r,X), N_X.itemset := X$;
6.      if $|X| = 1$ then let $N_X.prunedby := X$;
7.         else let $N_X.prunedby := FindDisjRule(X, Sup(r,X), \mathcal{F}req_{|X|-1} \cup \mathcal{F}req_{|X|-2})$;
8.      fi
9.   od

10.  let $i := 1$;
11.  while $\mathcal{F}req_i \neq \emptyset$ do
12.     let $\mathcal{C}_{i+1} := APRIORIGen(\mathcal{F}req_i)$;
13.     for all $X \in \mathcal{C}_{i+1}$ do
14.        Find node $N$ in $\mathcal{F}req_i$ such that $N.itemset \subset X$ and $N.prunedby \neq \emptyset$;
15.        if such $N$ exists then
16.           case $|N.prunedby|$ of :
17.           2:
18.              let $A, B$ be items such that $N.prunedby = \{A, B\}$;
19.              Find nodes $N_A, N_B$ in $\mathcal{F}req_i$ and $N_{AB}$ in $\mathcal{F}req_{i-1}$ such that
                   $N_A.itemset = X \setminus \{A\}$ and $N_B.itemset = X \setminus \{B\}$ and
                   $N_{AB}.itemset = X \setminus \{A, B\}$;
20.              let $S := N_A.support + N_B.support - N_{AB}.support$;
21.           1:
22.              Find node $N_A$ in $\mathcal{F}req_i$ such that $N_A.itemset = X \setminus N.prunedby$;
23.              let $S := N_A.support$;
24.           end case
25.           if $S \geq \sigma$ then
26.              Create new node $N_X$ in $\mathcal{F}req_{|X|}$ with $N_X.support := S$, $N_X.itemset := X$,
                   $N_X.prunedby := N.prunedby$;
27.           fi
28.        fi
29.     od
30.     let $i := i + 1$;
31.  od
32.  output $\{\langle N.itemset, N.support\rangle | N \in \bigcup_{j<i} \mathcal{F}req_j\}$;
```

In lines 1-9, Algorithm RegenFreq loads the input and converts it to a collection of nodes, one node per frequent itemset. The nodes corresponding to frequent itemsets of the negative border are marked by a nonempty content of the field *prunedby*.

In line 11, the algorithm enters a loop corresponding to increasing sizes of regenerated frequent itemsets. Within the i^{th} iteration of the loop, the algorithm uses $\mathcal{F}req_i$ and $\mathcal{F}req_{i-1}$ (the collections of the nodes corresponding to frequent itemsets of size i and $i-1$) to produce $\mathcal{F}req_{i+1}$ (i.e. frequent itemsets of size $i+1$). Candidate itemsets, i.e. itemsets of size $i+1$ having all their proper subsets frequent, are computed in line 12 and stored in $\mathcal{C}_{i+1}$. An additional test is performed in lines 14-15[3]. The condition ensures that at least one proper subset of the current candidate itemset is not disjunction-free. Other-

[3] Since APRIORIGen already accesses all nearest proper subsets of each candidate, then in the implementation of the algorithm the condition used in line 14 is checked within APRIORIGen.

wise (i.e., when all proper subsets are disjunction-free), the candidate can be skipped. Two cases fit skipped candidates: the candidate is not frequent, then we do not want to regenerate it anyway, or the candidate itemset is frequent. A frequent itemset of size $i+1$, for which all subsets of size i are frequent and disjunction-free, is either in $FreqDFree(r, \sigma)$ (when it is disjunction-free) or in $Freq\mathcal{B}d^{-}(r, \sigma)$ (non-disjunction-free itemset). In both cases, it is in the DBC, and thus the corresponding node has already been created in lines 1-9.

In lines 16-24, for each retained candidate itemset X, RegenFreq searches the information necessary to restore its support. RegenFreq finds a valid simple disjunctive rule leading to an equation that can be used to derive the support of X from the support of some of its subsets. Then, if $Sup(r, X) \geq \sigma$, RegenFreq creates a node N_X and fills it with the corresponding information (line 26). As previously, the value stored in $N_X.prunedby$ corresponds to any of the valid rules based on X, if there are more than one such rule.

Example 8 *Let us consider as input of* RegenFreq *the DBC given in Example 7 and show the most important steps of the execution of* RegenFreq *with $\sigma = 1$.*

Since the smallest frequent itemset of the negative border has 2 items, only the candidates of size 3 can satisfy the condition of the test performed in lines 14-15. Therefore, we start tracing the program at the second iteration (corresponding to candidates of size 3) of the main loop. Let us represent any node N by a triple $\langle N.itemset, N.support, N.prunedby \rangle$ Then $\mathcal{F}req_2$ is the following collection of triples: $\{\langle \{A, B\}, 2, \emptyset \rangle$, $\langle \{A, C\}, 3, \emptyset \rangle$, $\langle \{A, D\}, 2, \emptyset \rangle$, $\langle \{B, C\}, 4, \emptyset \rangle$, $\langle \{B, D\}, 2, \emptyset \rangle$, $\langle \{C, D\}, 2, \{C, D\} \rangle\}$.

$\mathcal{C}_3$ is $\{\{A, B, C\}, \{A, B, D\}, \{A, C, D\}, \{B, C, D\}\}$. $\{C, D\}$ is the only itemset with a nonempty prunedby field for the corresponding node, so the test in line 15 succeeds only for $X = \{A, C, D\}$ and $X = \{B, C, D\}$.

Using the nodes corresponding to $\{A, C\}$, $\{A, D\}$ and $\{A\}$, the support of $\{A, C, D\}$ is computed ($S = 1$, in line 20) and leads to the creation of the node corresponding to $\{A, C, D\}$ in line 26. Using the nodes corresponding to $\{B, C\}$, $\{B, D\}$ and $\{B\}$, leads similarly to the creation of the node corresponding to $\{B, C, D\}$ ($S = 1$).

The node corresponding to $\{A, B, C\}$ has already been created since $\{A, B, C\}$ is in the input $\mathcal{B}$, and thus $\mathcal{F}req_3$ is the following collection of triples: $\{\langle \{A, B, C\}, 2, \{C\} \rangle$, $\langle \{A, C, D\}, 1, \{C, D\} \rangle$, $\langle \{B, C, D\}, 1, \{C, D\} \rangle\}$. After the next iteration, $\mathcal{C}_4$ is empty, as well as $\mathcal{F}req_4$, so the program exits the loop and finishes.

Theorem 3 (Correctness of RegenFreq**)** *The algorithm* RegenFreq *outputs all and only σ-frequent itemsets along with their supports.*

Proof. By Lemma 1, if X is σ-frequent all its subsets are σ-frequent. Therefore, the proof is made by induction on $|X|$.

Hypothesis. Suppose that for every frequent itemset $X \neq \emptyset$, the algorithm RegenFreq correctly constructs the nodes corresponding to all proper subsets of X, notably that they are all present in their respective collections $\mathcal{F}req_j$, and that their fields *itemset*, *support* and *prunedby* are correctly filled.

First, we consider a σ-frequent itemset X. We are going to show that it is in the output.

Let us consider the case where X belongs to $DBC(r, \sigma)$. Then, X is σ-frequent and it is in the input. Therefore the corresponding node N_X is created in line 2 or in lines 5-8 and is never removed. Therefore, it will be output in line 32.

Suppose now that X is frequent, but does not belong to the DBC (is neither in $FreqDFree(r, \sigma)$ nor in $Freq\mathcal{B}d^-(r, \sigma)$). Therefore X is not disjunction-free. Moreover, since X is σ-frequent, but not in $Freq\mathcal{B}d^-(r, \sigma)$, there exists $Y \subset X$, such that $|Y| = |X| - 1$ and Y is not disjunction-free. The empty itemset is disjunction-free, thus $|Y| \geq 1$ and $|X| \geq 2$. Since the nodes corresponding to all subsets of X are correctly created (by *Hypothesis*), there exists a node N corresponding to Y (line 15) and N has a nonempty, correct value of *prunedby*.

By the same hypothesis, the nodes corresponding to all proper subsets of X are present in their respective collections $\mathcal{F}req_j$. Since $|X| \geq 2$, X is produced as candidate itemset by APRIORIGen in line 12 and it is considered in lines 16-27. The corresponding node is actually created in line 26, because X is σ-frequent. It will be in the output, because a created node is never removed.

For the equivalence, suppose that the output contains X along with its support. We are going to show that X is σ-frequent.

Observe that nodes are created in lines 2, 5-8 and 26. If the node N_X with $N_X.itemset = X$ is created in line 2 or in lines 5-8, then X belongs to the DBC, and thus is σ-frequent. If the node N_X is created in line 26, X is σ-frequent, because the corresponding value S is checked against σ in line 25. S is the correct support of X according to the fact that X is not disjunction-free, to the valid simple disjunctive rule $Y \setminus \{A, B\} \Rightarrow A \vee B$ corresponding to the value of *N.prunedby*, to Lemma 5 and to *Hypothesis* (*N.prunedby* is correct and the support inference equation is applied on correct values of subset supports).

Therefore, if the itemset X is in the output, it is σ-frequent. □

5 Comparing the representation size

In this section, we report a comparison between sizes of different representations. For different data sets and support thresholds we consider the collection of all frequent itemsets, the collection of all frequent closed sets [12] and the DBC.

Like the DBC, frequent closed sets can be also seen as a condensed representation of all frequent itemsets. It has been previously investigated in the literature and we will recall its definition and some properties when needed in Section 7. Here, we simply consider the size of this representation and we only note that it also allows regenerating all frequent itemsets along with their supports.

The three representations (frequent itemsets, frequent closed sets and the DBC) are composed of a collection of itemsets and their associated supports. We retain two measures that reflect the size of these representations:

- total number of itemsets in the representation, and
- flat-storage space used.

The first measure is interesting in itself, but it does not correspond directly to the amount of data to be stored. The benefit of the second measure is to characterize this aspect more precisely.

We consider the storage in a binary file without any ad-hoc encoding or compression, and we make the following assumptions. The information relative to an itemset can be stored using one integer for its support and one integer per item, all used integers are ranging from 0 to 67 557. We neglect the additional marker of the end of the sequence of items. And finally, we consider that the elementary storage unit is simply one 32-bit long integer in a fixed format. So, let $\mathcal{S}$ be the collection of itemsets in one of the three representations, the corresponding flat storage space (in bytes) is computed as $4 * \sum_{X \in \mathcal{S}}(|X| + 1)$.

We report representation sizes on three different commonly used data sets: *Mushroom* (characteristics of some mushroom species), *Connect-4* (collection of game-related state information), and *Pumsb* (PUMS census data). All these data sets have been preprocessed by researchers from IBM Almaden Research Center [4] . The particularity of the selected data sets is that they are very dense and the combinatorial explosion of the number of frequent itemsets makes the mining of all frequent itemsets together with their supports intractable for low support thresholds [19].

[4] http://www.almaden.ibm.com/cs/quest/data/long_patterns.bin.tar

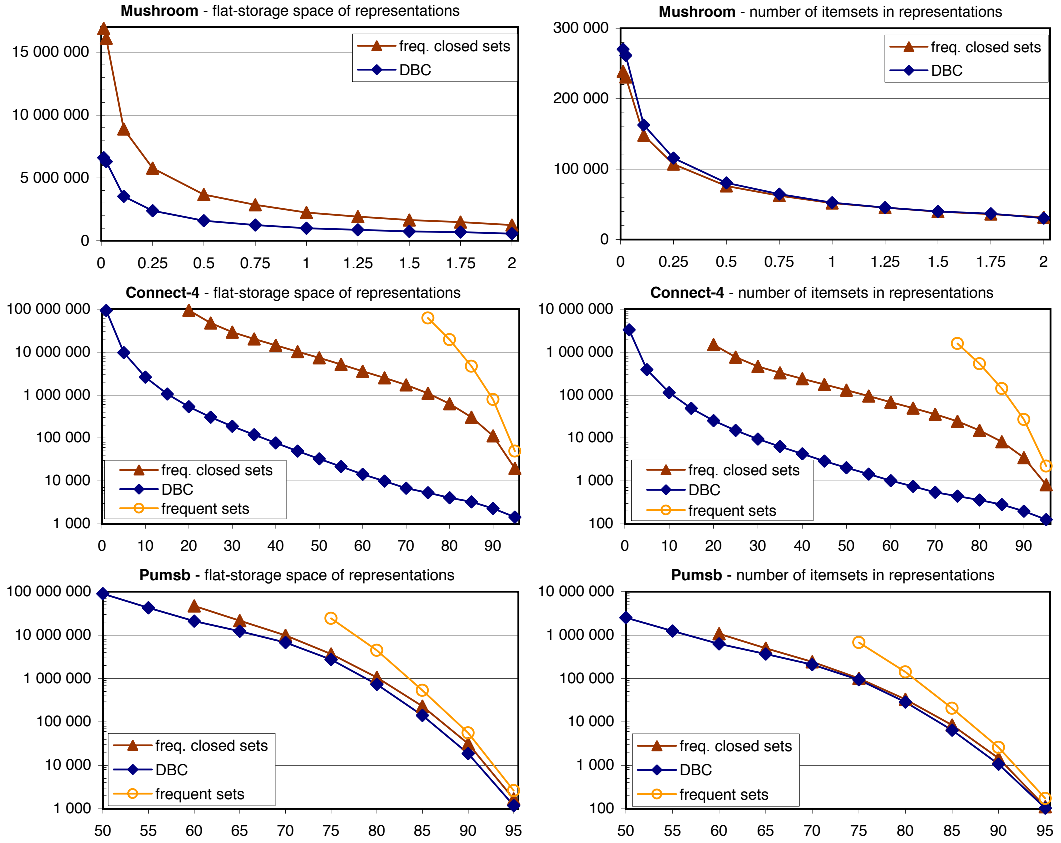


Fig. 3. Different representation sizes on the three data sets.

The representation sizes for several support thresholds are given in Figure 3 (note that some axes are logarithmically scaled). Lower values correspond to more condensed representations. It should be noticed that the support thresholds are given as relative frequency thresholds (i.e., 100 × absolute support threshold / total number of rows in the data set).

The *Mushroom* data set is based on 119 items only and contains 8124 rows — a relatively small number in data mining. Nonetheless, at the thresholds used in the experiments (Figure 3, upper graphics) the collection of all frequent itemsets was huge (more than 22 millions for a relative support threshold of 2%) and we could not extract frequent itemsets completely for thresholds lower than 2%, even with a non-naïve implementation of the APRIORI algorithm [2].

The *Connect-4* data set contains 67 557 rows, but a relatively small number of items (129). Its difficulty lies in a very high correlation. A huge number of frequent itemsets was observed even for as high frequency thresholds as 75% (Figure 3, middle graphics).

The last data set, *Pumsb*, contains 49 046 rows and is very challenging because of the high number of items (7117). For this reason, when decreasing the frequency threshold, the sizes quickly grow for all representations (see Figure 3, lower graphics).

In the experiments reported on Figure 3 in the left graphics, we can see that the DBC representation is smaller than the frequent closed sets representation when we consider their flat-storage spaces. The same holds with few exceptions if we count the number of itemsets required by each representation (Figure 3, right graphics).

For the comparison of the size of the DBC with the size of the collection of all frequent itemsets, according to Theorem 2 we know that the DBC is always smaller or equal. In the experiments presented in Figure 3, the difference is very important and ranges from one to four orders of magnitude.

6 Discovering the DBC

In this section we describe two algorithms to mine the DBC. Two main strategies have been proposed to explore the search space during frequent itemset mining: breadth-first (e.g., [2]) and depth-first [6,5]. Each of them has its pros and cons, and even if we consider only the extraction time criterion there is no always-winning strategy as we will show in Section 6.3. So, in this section we consider both strategies and we describe the corresponding algorithms. For each, we give an abstract version to find $FreqDFree(r, \sigma)$ and $Freq\mathcal{B}d^{-}(r, \sigma)$, and then we describe the key implementation issues.

The *anti-monotonicity* of disjunction-freeness w.r.t. itemset inclusion is important for an efficient mining (Lemma 6). Actually, the combined property "frequent disjunction-free" is used. It is also anti-monotone as stated by the following lemma.

Lemma 7 *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in FreqDFree(r, \sigma)$ then $Y \in FreqDFree(r, \sigma)$.*

Proof. Suppose that $X \in FreqDFree(r, \sigma)$. Let us consider any $Y \subseteq X$.

Since $X \in FreqDFree(r, \sigma)$, $X \in Freq(r, \sigma)$ and $X \in DFree(r)$. According to the anti-monotonicity of support (stated by Lemma 1), $Y \in Freq(r, \sigma)$. Similarly, the anti-monotonicity of disjunction-freeness (stated by Lemma 6) implies that $Y \in DFree(r)$.

Finally, $Y \in Freq(r, \sigma) \cap DFree(r) = FreqDFree(r, \sigma)$. □

During the extraction of the DBC, according to the contrapositive of this property, when an itemset is not frequent disjunction-free then there is no need to consider any of its proper supersets. This pruning criterion will be applied in breadth-first and depth-first strategies.

6.1 Breadth-first extraction

6.1.1 Abstract algorithm

The algorithm, presented below, is formulated as an instance of the generic levelwise-search algorithm[5] presented in [18]. It explores iteratively the itemset lattice (w.r.t. set inclusion) levelwise, starting from the empty itemset and stopping at the level of the largest itemset from $DBC(r, \sigma)$, or, less commonly, one level further. At each iteration, it scans the database to find which itemsets of the current level are frequent disjunction-free sets. Then, it generates candidates for the next iteration considering only the itemsets of the next level for which all proper subsets are frequent disjunction-free sets.

Algorithm 4 (HLinEx)

Input: *r a binary database over a set of items R, and σ an absolute support threshold.*

Output: *$FreqDFree(r, \sigma)$ and frequent itemsets of the negative border.*

```
let $\mathcal{C} := \{\emptyset\}$, $\mathcal{FDF} := \emptyset$, $\mathcal{FB} := \emptyset$, $i := 0$;
while $\mathcal{C} \neq \emptyset$ do
    Scan $r$ and compute supports of itemsets in $\mathcal{C}$;
    $\mathcal{FDF} := \mathcal{FDF} \cup \{X | X \in \mathcal{C}$ and $X$ is $\sigma$-frequent disjunction-free in $r\}$;
    $\mathcal{FB} := \mathcal{FB} \cup \{X | X \in \mathcal{C}$ and $X$ is $\sigma$-frequent and not disjunction-free in $r\}$;
    $\mathcal{C} := \{X | X \subseteq R$ and $|X| = i + 1$ and $\forall\, Y \subset X, |Y| = i \Rightarrow Y \in \mathcal{FDF}\}$;
    let $i := i + 1$;
od
output $\langle \mathcal{FDF}, \mathcal{FB} \rangle$;
```

Example 9 *Consider the binary database depicted in Figure 1. The execution of the algorithm* HLinEx *on this database with $\sigma = 1$ can be sketched as follows:*

Initialization
$\mathcal{C} := \{\emptyset\}$, $\mathcal{FDF} := \emptyset$, $\mathcal{FB} := \emptyset$
Iteration 1
$\mathcal{FDF} := \mathcal{FDF} \cup \{\emptyset\}$
$\mathcal{FB} := \mathcal{FB} \cup \emptyset$
$C := \{\{A\}, \{B\}, \{C\}, \{D\}\}$
Iteration 2
$\mathcal{FDF} := \mathcal{FDF} \cup \{\{A\}, \{B\}, \{C\}, \{D\}\}$
$\mathcal{FB} := \mathcal{FB} \cup \emptyset$
$C := \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}\}$
Iteration 3
$\mathcal{FDF} := \mathcal{FDF} \cup \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\}$

[5] Efficient frequent itemset mining algorithms like APRIORI [2] can also be seen as instances of this generic algorithm.

$\mathcal{FB} := \mathcal{FB} \cup \{\{C, D\}\}$
$C := \{\{A, B, C\}, \{A, B, D\}\}$

Iteration 4
$\mathcal{FDF} := \mathcal{FDF} \cup \emptyset$
$\mathcal{FB} := \mathcal{FB} \cup \{\{A, B, C\}\}$
$\mathcal{C} := \emptyset$

Output
$\langle \{\emptyset, \{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\},$
$\{\{C, D\}, \{A, B, C\}\}\rangle$.

Using the anti-monotonicity of the combined property stated by Lemma 7 and the correctness result of the levelwise search algorithm of [18], the following theorem is straightforward.

Theorem 4 (Correctness of HLinEx) *Given r a binary database over a set of items R, and σ an absolute support threshold, Algorithm 4 computes $\langle \mathcal{FDF}, \mathcal{FB}\rangle$, and we have $FreqDFree(r, \sigma) = \mathcal{FDF}$ and $Freq\mathcal{B}d^{-}(r, \sigma) = \mathcal{FB}$.*

6.1.2 Implementation issues

Techniques similar to the ones presented in [2] for levelwise mining of frequent itemsets are used. The candidate generation (line 6) is made using a join-based APRIORIGen function (recalled in Section 4), the data set is linearly scanned to count supports of candidates (line 3) and the itemset support counters are updated w.r.t. a row of the data set using a *prefix-tree* data structure.

Implementing line 6 as a call to APRIORIGen function requires the size of itemsets of the current level to be at least 1. Therefore, the empty itemset is processed apart and the first iteration of the loop starts with singleton itemsets as candidates. Moreover, APRIORIGen uses only σ-frequent disjunction-free sets of size i to generate candidates of size $i+1$. Therefore, we partition $\mathcal{FDF}$ into $\mathcal{FDF}_i$ according to itemset size, and use only the portion corresponding to itemsets of size i for the call.

A specific aspect is the implementation of the disjunction-freeness test (lines 4 and 5). When we need to test this property for an itemset X, we already know the supports of all its subsets. So we check if there exist A and B, items of X, such that $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$. By Lemma 5 and Definition 6, X is disjunction-free if and only if no such a pair of items exists. Observe, that the function FindDisjRule (given in Section 4) realizes this test for X using supports of its proper subsets of size $|X| - 1$ and $|X| - 2$. Since all these proper subsets of X are already in $\mathcal{FDF}_{|X|-1}$ and $\mathcal{FDF}_{|X|-2}$ when the test is to be performed for X in lines 4 and 5, we can simply call $FindDisjRule(X, Sup(r, X), \mathcal{F}DF_{|X|-1} \cup \mathcal{F}DF_{|X|-2})$.

6.2 Depth-first extraction

6.2.1 Abstract algorithm

The algorithm given below is described by means of a recursive function $Find$ which explores the itemset lattice in a depth-first manner. A call $Find(X, r, \sigma, Tail)$ finds in r the σ-frequent disjunction-free sets in a well-defined partition of the search space. The algorithm uses the efficient divide-and-conquer strategy of [6,5] as follows. A call to the function will consider only the partition of the search space corresponding to proper supersets of X. Moreover, this subspace is further reduced to subsets of $X \cup Tail$, i.e. proper supersets of X that have not been examined earlier (see Algoritm 5).

Considering only supersets of X enables an efficient determination of support and of disjunction-freeness, because, only a restricted part of the database is required. Moreover, this restricted part of the database is further restricted in a recursive manner for recursive calls. Thus, in case of most calls to it, $Find$ works on a relatively small fraction of the database.

To identify which are the rows necessary for testing disjunction-freeness, we will recall the function FindDisjRule. For testing disjunction-freeness of the itemset X, FindDisjRule needs the supports of some subsets of X. Not of all subsets however — only the subsets of X with up to 2 items missing are required. Therefore, for the abstract version of the algorithm, we consider the following function, denoted $\mathcal{A}ug\mathcal{M}(r, X)$, which selects the rows of a database such that the supports of subsets of X of size at least $|X|-2$ remain preserved.

Definition 11 Given r a binary database over R and an itemset X, the augmented set of rows matching X is defined as $\mathcal{A}ug\mathcal{M}(r, X) = \{t \in r | \ |X \setminus t| \leq 2\}$.

Lemma 8 *Let r be a binary database over R, and X, Y be sets of items such that $Y \subseteq X \subseteq R$. Then $\mathcal{A}ug\mathcal{M}(r, X) \subseteq \mathcal{A}ug\mathcal{M}(r, Y)$.*

Proof. Let t be any row in r that belongs to $\mathcal{A}ug\mathcal{M}(r, X)$. From Definition 11, a row t belongs to $\mathcal{M}(r, X)$ iff $|X \setminus t| \leq 2$. $Y \subseteq X$ implies that $Y \setminus t \subseteq X \setminus t$. It follows that $|Y \setminus t| \leq |X \setminus t| \leq 2$ and further that $t \in \mathcal{A}ug\mathcal{M}(r, Y)$. Thus, $\mathcal{A}ug\mathcal{M}(r, X) \subseteq \mathcal{A}ug\mathcal{M}(r, Y)$. □

Algorithm 5 (VLinEx) *Function $Find(X, r, \sigma, Tail)$*

Input: *X the itemset considered as the current starting point in the search* *space, r a binary database over a set of items R, σ an absolute support threshold, and $Tail$ a set of items such that the itemsets X and $X \cup Tail$ delimit the partition of the search-space to be considered by this call to the function.*

Output: *The elements of $FreqDFree(r, \sigma)$ that are proper supersets of X and subsets of $X \cup Tail$, and a set of additionally explored frequent itemsets (a superset of the collection of the frequent itemsets of the negative border).*

```
let $\mathcal{C} := \{X \cup \{A\} | A \in Tail\}$;
let $\mathcal{FDF} := \emptyset$, $\mathcal{FB} := \emptyset$;
Scan $r$ and compute supports of itemsets in $\mathcal{C}$;
for all $A \in Tail$ in the ascending order given by $\prec$ do
    let $Tail := Tail \setminus \{A\}$;
    let $Y := X \cup \{A\}$;   // $Y \in \mathcal{C}$
    if $Y$ is a $\sigma$-frequent disjunction-free set in $r$ then
        let $\langle \mathcal{FDF}', \mathcal{FB}' \rangle := Find(Y, \mathcal{A}ug\mathcal{M}(r, Y), \sigma, Tail)$;
        let $\mathcal{FDF} := \mathcal{FDF} \cup \mathcal{FDF}' \cup \{Y\}$;
        let $\mathcal{FB} := \mathcal{FB} \cup \mathcal{FB}'$;
    elsif $Y$ is a $\sigma$-frequent in $r$ then
        let $\mathcal{FB} := \mathcal{FB} \cup \{Y\}$;
    fi
od
output $\langle \mathcal{FDF}, \mathcal{FB} \rangle$;
```

It should be noticed that the algorithm does not consider the empty itemset, and also that it does not provide an immediate characterization of the negative border. However, the empty itemset can be handled trivially and the computation of the exact collection of frequent itemsets of the negative border can be performed in a straightforward post-processing step using $\mathcal{FB}$ — non-minimal itemsets (w.r.t. set inclusion) of this collection must simply be removed to obtain the collection of frequent itemsets of the negative border.

Example 10 *Let us consider again the binary database depicted in Figure 1. Suppose that the linear order $\prec$ over the set of items is simply $A \prec B \prec C \prec D$. A call to the function $Find$ on this database with $\sigma = 1$, $X = \emptyset$ and $Tail = \{A, B, C, D\}$ leads to the following recursive calls:*

```
        $\mathcal{C} = \{\{A\}, \{B\}, \{C\}, \{D\}\}$
        since $\{A\}$ is frequent and disjunction-free
        then call $Find$ with $X = \{A\}$ and $Tail = \{B, C, D\}$
1.          $\mathcal{C} = \{\{A, B\}, \{A, C\}, \{A, D\}\}$
            since $\{A, B\}$ is frequent and disjunction-free
            then call $Find$ with $X = \{A, B\}$ and $Tail = \{C, D\}$
1.1             $\mathcal{C} = \{\{A, B, C\}, \{A, B, D\}\}$
                $\{A, B, C\}$ is frequent but not disjunction-free
                $\{A, B, D\}$ is not frequent
                output $\langle \emptyset, \{A, B, C\} \rangle$
            since $\{A, C\}$ is frequent and disjunction-free
            then call $Find$ with $X = \{A, C\}$ and $Tail = \{D\}$
1.2             $\mathcal{C} = \{\{A, C, D\}\}$
```

```
            ${A, C, D}$ is frequent but not disjunction-free
            output $\langle \emptyset, \{A, C, D\} \rangle$
        since $\{A, D\}$ is frequent and disjunction-free
        then call Find with $X = \{A, D\}$ and $Tail = \emptyset$
1.3         $\mathcal{C} = \emptyset$
            output $\langle \emptyset, \emptyset \rangle$
        $\mathcal{F}DF$ collected is $\{\{A, B\}, \{A, C\}, \{A, D\}\}$
        $\mathcal{F}B$ collected is $\{\{A, B, C\}, \{A, C, D\}\}$
        output $\langle \mathcal{F}DF, \mathcal{F}B \rangle$
    since $\{B\}$ is frequent and disjunction-free
    then call Find with $X = \{B\}$ and $Tail = \{C, D\}$
2.      $\mathcal{C} = \{\{B, C\}, \{B, D\}\}$
        since $\{B, C\}$ is frequent and disjunction-free
        then call Find with $X = \{B, C\}$ and $Tail = \{D\}$
2.1         $\mathcal{C} = \{\{B, C, D\}\}$
            $\{B, C, D\}$ is frequent but not disjunction-free
            output $\langle \emptyset, \{B, C, D\} \rangle$
        since $\{B, D\}$ is frequent and disjunction-free
        then call Find with $X = \{B, D\}$ and $Tail = \emptyset$
2.2         $\mathcal{C} = \emptyset$
            output $\langle \emptyset, \emptyset \rangle$
        $\mathcal{F}DF$ collected is $\{\{B, C\}, \{B, D\}\}$
        $\mathcal{F}B$ collected is $\{\{B, C, D\}\}$
        output $\langle \mathcal{F}DF, \mathcal{F}B \rangle$
    since $\{C\}$ is frequent and disjunction-free
    then call Find with $X = \{C\}$ and $Tail = \{D\}$
3.      $\mathcal{C} = \{\{C, D\}\}$
        $\{C, D\}$ is frequent but not disjunction-free
        output $\langle \emptyset, \{C, D\} \rangle$
    since $\{D\}$ is frequent and disjunction-free
    then call Find with $X = \{D\}$ and $Tail = \emptyset$
4.      $\mathcal{C} = \emptyset$
        output $\langle \emptyset, \emptyset \rangle$
    $\mathcal{F}DF$ collected is $\{\{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\}$
    $\mathcal{F}B$ collected is $\{\{C, D\}, \{A, B, C\}, \{A, C, D\}, \{B, C, D\}\}$
    output $\langle \mathcal{F}DF, \mathcal{F}B \rangle$
```

When compared to the output obtained with HLinEx *in Example 9, $\mathcal{F}B$ contains non-minimal elements. As mentioned in the previous paragraph, these elements can be removed by a simple post-processing step.*

The correctness of Algorithm 5 is stated by the following theorem.

Theorem 5 (Correctness of VLinEx**)** *Given r a binary database over a set of items R, and σ an absolute support threshold, the call $Find(\emptyset, r, \sigma, R)$ returns $\langle \mathcal{F}DF, \mathcal{F}B \rangle$, and we have $FreqDFree(r, \sigma) \setminus \{\emptyset\} = \mathcal{F}DF$ and $Freq\mathcal{B}d^{-}(r, \sigma) \subseteq \mathcal{F}B$.*

Proof. First of all, we recall that the list $orderedList(X)$ is the permutation of all items in X that is sorted in ascending order according to the linear order $\prec$ for items. $X[i]$ designates i^{th} element of $orderedList(X)$.

Let us now consider $\langle \mathcal{FDF}, \mathcal{FB} \rangle$ returned by $Find(\emptyset, r, \sigma, R)$.

Below, we show that $\forall Y, Y \in FreqDFree(r, \sigma) \setminus \{\emptyset\} \Rightarrow Y \in \mathcal{FDF}$ using an induction on $|Y|$.

Let $Y \in FreqDFree(r, \sigma)$ and $|Y| = 1$ (i.e. Y is a singleton itemset). Note that every singleton itemset is considered directly by $Find(\emptyset, r, \sigma, R)$ (line 6) and therefore if Y is σ-frequent disjunction-free (line 7) we have $Y \in \mathcal{FDF}$ (line 9).

Induction hypothesis. Let Z be an itemset such that $|Z| \geq 2$. Suppose that the property holds for every itemset $X \subset Z$ such that $orderedList(X)$ is a prefix of $orderedList(Z)$.

Given $Y \in FreqDFree(r, \sigma)$, such that $|Y| = n+1 \geq 2$, let x be an n-element prefix of $orderedList(Y)$ and X be the set of items occurring in x. Note that $Y[n+1]$ is the item marking off the difference between Y and X.

By Lemma 7, $X \in FreqDFree(r, \sigma)$, and then, by the induction hypothesis, $X \in \mathcal{FDF}$. Adding an itemset to $\mathcal{FDF}$ for the first time in line 9 is preceded by a call to $Find$ (line 8) with that itemset as the first parameter. It follows that $Find$ has been called at least once using $Find(X, r_X, \sigma, Tail_X)$, where r_X and $Tail_X$ are the corresponding instances of parameters r and $Tail$, at the time $Find$ is called.

Similarly, for every $k = \{1, \ldots, n\}$, $Find(\{X[1], X[2], \ldots, X[k]\}, \ldots)$ has been called in line 8. At any of these events, for every item B in the corresponding instance of $Tail$ such that $X[k] \prec B$, B had not yet been processed (due to the ascending order in which items are processed in line 4). This implies that B is not yet removed from the corresponding $Tail$ in line 5.

Note that for all $k = \{1, \ldots, n\}$ the property $X[k] \prec Y[n+1]$ holds. Otherwise, the list $orderedList(Y)$ would not have followed the ascending order of items, which would have contradicted with how $orderedList(Y)$ is defined.

Combining the last two properties proves that $Y[n+1]$ is in $Tail_X$.

Let us focus on the call $Find(X, r_X, \sigma, Tail_X)$. This call starts by the following candidate generation step $\mathcal{C} := \{X \cup \{A\} | A \in Tail_X\}$. We know that $Y[n+1] \in Tail_X$. Therefore, Y is generated as candidate (in line 1) and tested for sufficient support and disjunction-freeness (in line 7). Since $Y \in FreqDFree(r, \sigma)$, Y is collected in $\mathcal{FDF}$.

The soundness of the algorithm (i.e., $\forall Y, Y \in \mathcal{FDF} \Rightarrow Y \in FreqDFree(r, \sigma)$) is immediate (lines 7 and 9).

Finally, we consider again the call $Find(\emptyset, r, \sigma, R)$, which returns $\langle \mathcal{FDF}, \mathcal{FB} \rangle$. Let Y be an element of $Freq\mathcal{B}d^{-}(r, \sigma)$ such that $|Y| = n + 1$, and x be the n-element prefix of $orderedList(Y)$. Let X be the set of items occurring in x.

By Definition 8, $X \in FreqDFree(r, \sigma)$, and since $\forall Z, Z \in FreqDFree(r, \sigma) \setminus \{\emptyset\} \Rightarrow Z \in \mathcal{FDF}$, we have $X \in \mathcal{FDF}$ unless $X = \emptyset$. Using the same reasoning as above we know that Y is generated as candidate in line 1, thus tested in lines 7 and 11, and stored in line 12. Thus $Freq\mathcal{B}d^{-}(r, \sigma) \subseteq \mathcal{FB}$. □

6.2.2 Implementation issues

We combined best features from state-of-the-art algorithms implementing a depth-first strategy for frequent itemset mining. In particular, we use a support counting technique similar to the one presented in [5], and a compact storage of a collection of rows (instances of the parameter r of $Find$) in a prefix-tree structure as described in [6].

An optimization specific to VLinEx consists in preserving only items of $Y \cup Tail$ in line 8 (when the selection of rows by $\mathcal{A}ug\mathcal{M}$ occurs), because the exploration of the corresponding partition of the search space neither contains other items nor requires supports involving them.

The loop of line 4 enumerates the items $A \in Tail$ following the ascending order of supports of the corresponding itemsets $X \cup \{A\}$ in r. This order of exploration has been inspired by [19] and [6]. It aims at some balancing of the size of the collection of rows passed to the nested call to $Find$. Indeed, as $Tail_X$ gets smaller and smaller by removing items in line 5, the collection of rows passed to $Find$ in line 8 preserves statistically less items for a single row. At the same time, as the number of rows matched by $X \cup \{A\}$ in r increases, $\mathcal{A}ug\mathcal{M}$ selects statistically more rows. Therefore, both effects oppose each other.

Considering items in this (changed) order in line 4 does not compromise the correctness of the algorithm, since a candidate itemset is still generated at least once.

Finally, for an itemset Y, the test of the disjunction-freeness is performed as follows. We consider all pairs of items A and B in Y, and compute $GenSup(r, (Y \cup \{\overline{A}, \overline{B}\}) \setminus \{A, B\})$. If this support is equal to zero for one of the pairs, then, by Lemma 3, Y is not disjunction-free in r.

Further details about the implementation issues may be found in [20].

In Section 5, we experimentally compared the size of the condensed representation based on frequent closed sets to the size of the DBC. In this section, we consider their respective extraction times.

To mine efficiently the frequent closed sets we use two algorithms proposed recently: a breadth-first algorithm called CLOSE [12] and a depth-first algorithm called CLOSET [21]. Additionally, we report the extraction times of the well-known APRIORI algorithm, which extracts all frequent itemsets. We have implemented these algorithms as described by their authors. We notice however that the implementations of all algorithms (including HLinEx and VLinEx) use the same low-level data structures and techniques, in order to ensure a fair comparison. All prototypes have been implemented in C++, and a similar effort has been spent on specific fine-tuning of each of them. We have run all experiments on a PC with 256 MB of memory and an 800 MHz Pentium III processor under Linux operating system.

We compared the extraction times of frequent closed sets and of the DBC on the data sets that have been described in Section 5. We varied data sets and frequency thresholds in order to get a meaningful overview. The running times, in seconds, are given in Figure 4 (note that some axes are logarithmically scaled and that we use relative frequency thresholds, as in Section 5). The results are given for HLinEx, VLinEx, for the implementations of CLOSE and CLOSET, and additionally for the implementation of APRIORI.

On the *Mushroom* data set, we observed that extracting the DBC is advantageous (see Figure 4, upper graphics) especially using HLinEx. Observed speed-up was up to 5 times for breadth-first extractors and up to 8.5 times for depth-first extractors. With the extraction time above 1 hour, the curve corresponding to APRIORI is exceptionally not reported on the figure for this data set. Thus, we could magnify the difference between extraction times for frequent closed sets and the DBC.

The *Connect-4* data set is very difficult for mining frequent itemsets (see Figure 4, middle graphics), and mining directly frequent closed sets using CLOSE and CLOSET allows a significant improvement (compare the results for CLOSE with the ones for APRIORI). Further significant improvements (over CLOSE and CLOSET) can be achieved by using HLinEx and VLinEx. At lowest frequency thresholds for which we were able to extract frequent closed sets, we observed over 150 times faster extractions for HLinEx vs. CLOSE and up to 260 times faster extractions for VLinEx vs. CLOSET.

In the case of the last data set, i.e. of *Pumsb*, extracting frequent patterns was very difficult for all extractors, and CLOSE was not significantly advantageous

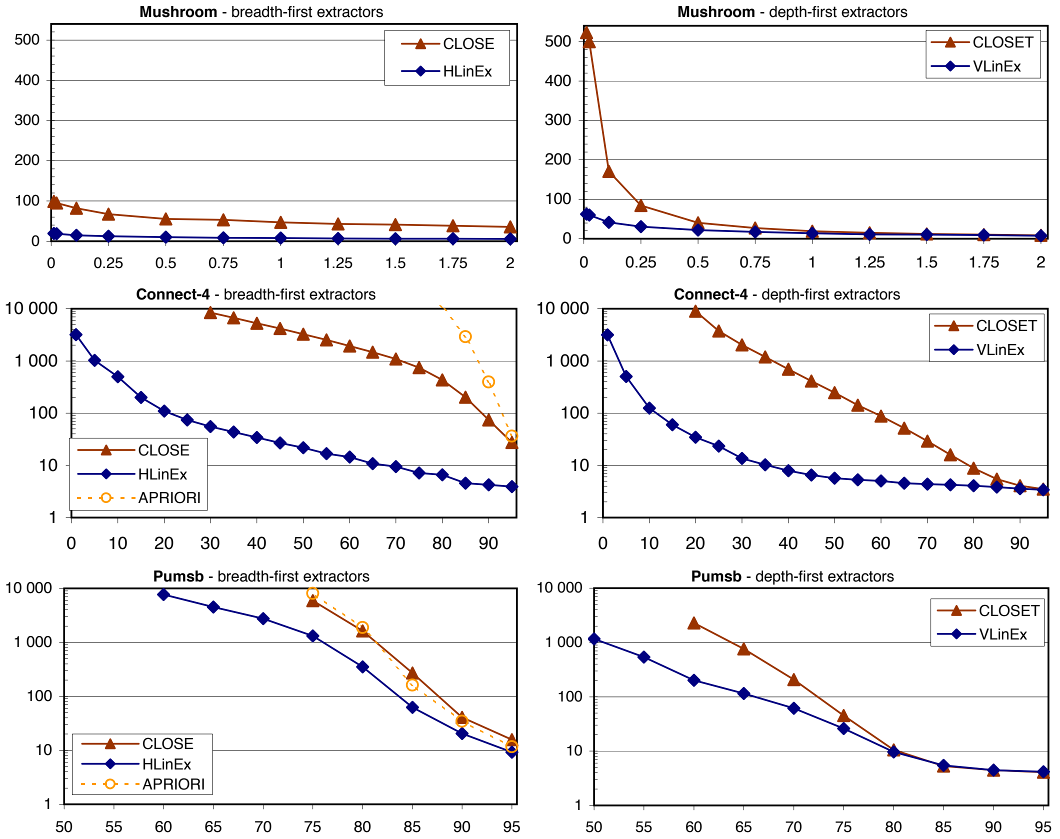


Fig. 4. Experiments on the three data sets with breadth-first (left) and depth-first (right) algorithms.

over APRIORI. However, HLinEx and VLinEx offer an evident benefit at lower frequency thresholds (see Figure 4, lower graphics).

In nearly all experiments on the three data sets, the extraction of the DBC is significantly more efficient than the extraction of the frequent closed sets. For the very difficult cases, the DBC can be extracted at lower frequency thresholds, using the same resources.

7 More about practical use of the DBC

The DBC is a condensed representation of frequent itemsets. The experiments presented in Sections 5 and 6 have shown that the DBC is much smaller than the collection of frequent itemsets and that it can be extracted efficiently, even in difficult cases. Thus, in practice, the DBC can be advantageously extracted and stored instead of frequent itemsets to derive important data mining patterns (most commonly *association rules* [22]).

We show now that the DBC is also an interesting condensed representation of the frequent closed sets and thus also offers the same practical benefits as the closed sets themselves. Among them we can mention the ability to generate directly rule covers in order to present to the expert a compact summary of (typically) huge collections of associations rules as described in [16].

In Section 5, we have already described the interest of the DBC w.r.t. closed sets in terms of representation size. In this section, we give an algorithm to convert the DBC into the collection of frequent closed sets. Then, we present experiments showing that in the most difficult cases it is more efficient to extract the DBC and to convert it into closed sets than to extract the closed sets directly.

First in Section 7.1, we recall some definitions and properties concerning closed sets and then give some additional preliminaries in Section 7.2.

7.1 Condensed representation based on frequent closed sets

We simply recall the basics of the closed sets representation. For details and proofs of the lemmas see [12].

We defined in Section 2 the multiset $\mathcal{M}(r, X)$ of rows matched by the itemset X in r. Now, we give the definition of $\mathcal{I}(r)$, the set of items matching a multiset of rows r.

Definition 12 ***(itemset matching a multiset of rows)*** Given r a binary database over R, $\mathcal{I}(r) = \{A \in R | \forall t \in r, A \in t\}$.

Example 11 *Intuitively the itemset matching a multiset of rows is composed of all items appearing in all rows.* $\mathcal{I}(\{\{A, D\}, \{A, B, D\}, \{A, C, D\}\})$ *is* $\{A, D\}$.

Definition 13 ***(closed itemset)*** Given r a binary database over R, X is a closed set w.r.t. r iff $X = \mathcal{I}(\mathcal{M}(r, X))$. $\mathit{Closed}(r) = \{X | X \subseteq R$ and $X = \mathcal{I}(\mathcal{M}(r, X))\}$ denotes the set of all closed itemsets in r.

Definition 13 states that an itemset X is closed w.r.t. r if and only if it contains every item that is present in all rows of r matched by X.

Example 12 *In the binary database depicted in Figure 1, the itemsets* $\{A\}$ *and* $\{B, C\}$ *are closed sets, whereas the itemset* $\{A, B\}$ *is not. Observe that the rows matching* $\{A, B\}$ *are all containing also the item* C*. So,* $\{A, B\} \neq \mathcal{I}(\mathcal{M}(r, \{A, B\})) = \{A, B, C\}$ *and thus* $\{A, B\}$ *is not closed.*

The compound function $\mathcal{I}(\mathcal{M}(r, X))$ returns the so-called *closure* of X w.r.t. r, and the corresponding operator is denoted in the following as $\mathcal{I} \circ \mathcal{M}$. Relevant properties of this operator are stated in the following lemmas.

Lemma 9 *Let r be a binary database over R, and X, Y be itemsets such that $X \subseteq Y \subseteq R$. $\mathcal{I} \circ \mathcal{M}(r, X) \subseteq \mathcal{I} \circ \mathcal{M}(r, Y)$.*

Lemma 10 *Let r be a binary database over R, and X be an itemset such that $X \subseteq R$. $\forall Y, X \subseteq Y \subseteq \mathcal{I} \circ \mathcal{M}(r, X) \Rightarrow Sup(r, Y) = Sup(r, X)$.*

Lemma 11 *Let r be a binary database over R, and X, Y be itemsets such that $X \subseteq Y \subseteq R$. $Sup(r, Y) = Sup(r, X)$ implies $Y \subseteq \mathcal{I} \circ \mathcal{M}(r, X)$.*

We can now define the condensed representation based on frequent closed sets, as used in this paper.

Definition 14 ***(frequent closed itemsets)*** Given r a binary database over R and σ a frequency threshold, the set of all σ-frequent closed itemsets w.r.t. r is defined as $FreqClosed(r, \sigma) = Freq(r, \sigma) \cap Closed(r)$.

Definition 15 ***(representation based on frequent closed itemsets)*** The condensed representation based on frequent closed itemsets is the collection of all σ-frequent closed sets w.r.t. a data set r along with the corresponding supports.

7.2 Concepts used in the conversion algorithm

In this section, we give some preliminary definitions and properties needed in Section 7.3 to present the conversion algorithm. In particular, we use the concept of *0-free set*, introduced in a different context in [13,14].

Informally, an itemset X is 0-free if there is no rule $Y \Rightarrow A$ with no exceptions such that $Y \subset X$ and $A \in X \setminus Y$. This notion is captured more precisely by two following definitions.

Definition 16 ***(simple consequence rule)*** Let X be a set of items, a *simple consequence rule* based on X is an expression of the form $Y \Rightarrow A$, where $Y \subset X$ and $A \in X \setminus Y$. Let r be a binary database over R, where $X \subseteq R$. The simple consequence rule $Y \Rightarrow A$ is *valid* in r (i.e., has no exceptions) if and only if $\mathcal{M}(r, Y) = \mathcal{M}(r, Y \cup \{A\})$.

Example 13 *$\{A, B\} \Rightarrow C$ is a simple consequence rule based on $\{A, B, C, D\}$ and is valid in the binary database of Figure 1.*

Definition 17 ***(0-free set)*** Given r a binary database over R, X is a *0-free set* w.r.t. r iff there is no valid simple consequence rule $Y \Rightarrow A$ based on X in r. The set of all 0-free sets w.r.t. r is noted $ZFree(r)$.

Example 14 *Let us consider again the binary database depicted in Figure 1. $\{A, C, D\}$ is a 0-free set since no valid simple consequent rule can be formed using only the items A, C and D.*

Anti-monotonicity of 0-freeness w.r.t. itemset inclusion follows directly from the definition of 0-free sets and is stated by the next lemma.

Lemma 12 *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in ZFree(r)$ then $Y \in ZFree(r)$.*

The following lemmas are needed in Section 7.3 to demonstrate the correctness of the conversion algorithm.

Lemma 13 *Let r be a binary database over R, X be an itemset such that $X \subseteq R$, A be a single positive item in $R \setminus X$ and $\overline{A}$ its corresponding negative item.*

Then the following three propositions are equivalent:
(1) $Sup(r, X \cup \{A\}) = Sup(r, X)$,
(2) $X \Rightarrow A$ is valid in r,
(3) $Gen\mathcal{M}(r, X \cup \{\overline{A}\}) = \emptyset$.

Proof. Observe that $\mathcal{M}(r, X) = \{t | t \in \mathcal{M}(r, X \cup \{A\}) \vee t \in Gen\mathcal{M}(r, X \cup \{\overline{A}\})\}$ and that the multisets $\mathcal{M}(r, X \cup \{A\})$ and $Gen\mathcal{M}(r, X \cup \{\overline{A}\})$ are mutually exclusive. Therefore $Sup(r, X) = Sup(r, X \cup \{A\}) + GenSup(r, X \cup \{\overline{A}\})$. Consequently, *(1)* is equivalent to $GenSup(r, X \cup \{\overline{A}\}) = 0$, and the latter to *(3)*. Finally, $\mathcal{M}(r, X) = \mathcal{M}(r, X \cup \{A\})$ $(= \{t | t \in \mathcal{M}(r, X \cup \{A\}) \vee t \in \emptyset\})$, which satisfies the definition of $X \Rightarrow A$ being valid in r (i.e. *(2)*). □

Lemma 14 *Given r a binary database over R, X is a 0-free set w.r.t. r iff $\forall Y \subset X, Sup(r, Y) > Sup(r, X)$.*

Proof. Let us prove the equivalence of negations.

First, suppose that there exists $Y \subset X$ such that $Sup(r, Y) \leq Sup(r, X)$. By Lemma 1, $Sup(r, Y) \geq Sup(r, X)$, so $Sup(r, Y) = Sup(r, X)$. Let $A \in X \setminus Y$. By the anti-monotonicity of support w.r.t. the itemset inclusion, $Y \subseteq X \setminus \{A\} \subset X$ implies $Sup(r, X \setminus \{A\}) = Sup(r, X)$. By Lemma 13, we deduce that $X \setminus \{A\} \Rightarrow A$ is valid, which proves that X is not a 0-free set w.r.t. r.

For the equivalence, suppose that X is not 0-free w.r.t. r, i.e. $\exists Y \subset X, \exists A \in X \setminus Y$ such that $Y \Rightarrow A$ is valid in r. By Lemma 13, $Y \Rightarrow A$ is valid in r implies that $Gen\mathcal{M}(r, Y \cup \{\overline{A}\}) = \emptyset$.

Since $Y \subseteq X \setminus \{A\}$ implies $Y \cup \{\overline{A}\} \subseteq (X \setminus \{A\}) \cup \{\overline{A}\}$, Lemma 2 points that $Gen\mathcal{M}(r, (X \setminus \{A\}) \cup \{\overline{A}\}) \subseteq Gen\mathcal{M}(r, Y \cup \{\overline{A}\}) = \emptyset$. Therefore, no rows belong to $Gen\mathcal{M}(r, (X \setminus \{A\}) \cup \{\overline{A}\})$.

Finally, by Lemma 13, $Sup(X \setminus \{A\}) = Sup(X)$, which proves that $\neg(\forall Y \subset X, Sup(r, Y) > Sup(r, X))$. □

Definition 18 ***(frequent 0-free sets)*** Given r a binary database over R and σ a frequency threshold, the set of all σ-frequent 0-free sets is defined as $FreqZFree(r, \sigma) = Freq(r, \sigma) \cap ZFree(r)$.

Definition 19 ***(frequent itemsets of the negative border of 0-free sets)*** Let r be a binary database over a set of items R. Frequent itemsets of the negative border of $FreqZFree(r, \sigma)$ is noted $FreqZFree\mathcal{B}d^-(r, \sigma)$ and is defined as follows: $FreqZFree\mathcal{B}d^-(r, \sigma) = \{X | X \subseteq R, X \notin FreqZFree(r, \sigma) \wedge (\forall Y \subset X, Y \in FreqZFree(r, \sigma))\} \cap Freq(r, \sigma)$.

7.3 Condensed representation conversion algorithm

In this section, we describe an algorithm that changes the DBC into the condensed representation based on frequent closed sets. Although a more abstract presentation is possible, we give a detailed version to ease the implementation of an efficient converter.

The algorithm is based on the following theorem.

Theorem 6 *Let r be a binary database over a set of items R, σ be a frequency threshold and Z be a σ-frequent 0-free set. Then, $\mathcal{I} \circ \mathcal{M}(r, Z) = Z_1 \cup Z_2 \cup Z_3 \cup Z_4$, where $Z_1, \ldots, Z_4$ are:*

$Z_1 = \{A \in R | Z \cup \{A\} \in FreqZFree\mathcal{B}d^-(r, \sigma) \wedge Sup(r, Z) = Sup(r, Z \cup \{A\})\}$

$Z_2 = \{A \in R | \exists X, |X| < |Z| \wedge X \in FreqZFree(r, \sigma) \wedge Sup(r, X) = Sup(r, Z) \wedge Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)\}$

$Z_3 = \{A \in R | A \in Z \vee \exists X, X \subset Z \wedge X \in FreqZFree(r, \sigma) \wedge A \in \mathcal{I} \circ \mathcal{M}(r, X))\}$

$Z_4 = \{A \in R | A \notin Z \wedge \exists B \in Z, (Z \cup \{A\}) \setminus \{B\} \in FreqZFree(r, \sigma) \wedge Sup(r, (Z \cup \{A\}) \setminus \{B\}) = Sup(r, Z) \wedge \exists X, X \subset (Z \cup \{A\}) \setminus \{B\}, B \in \mathcal{I} \circ \mathcal{M}(r, X)\}$

This theorem states that an item A is an element of the closure of a σ-frequent 0-free set Z if and only if it satisfies at least one of the following properties:

- A is such that the rule $Z \Rightarrow A$ is valid,
- $Z \cup \{A\}$ is included in the closure of a frequent 0-free set X having the same support as Z but a size strictly smaller than the size of Z,

- A is in Z or in the closure of a proper subset of Z,
- A is not in Z, but is in a 0-free set W such that firstly Z and W have the same size and support, secondly they contain the same items except one, and finally $Z \setminus W$ is included in the closure of a proper subset of W.

The proof of this theorem requires the following two lemmas.

Lemma 15 *Let r be a binary database over a set of items R, σ be a frequency threshold, Z be a σ-frequent 0-free set and $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$. If $A \notin Z_1$ then $\exists Y \subset Z \cup \{A\}, \exists B \in Y, Y \setminus \{B\} \in \mathit{FreqZFree}(r, \sigma) \wedge B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$.*

Proof. Let $A \in \mathcal{I} \circ \mathcal{M}(r, Z) \setminus Z_1$. $A \notin Z_1$ implies that $Z \cup \{A\} \notin \mathit{FreqZFree}\mathcal{B}d^-(r, \sigma) \vee \mathit{Sup}(r, Z) \neq \mathit{Sup}(r, Z \cup \{A\})$. But $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ implies that $\mathit{Sup}(r, Z) = \mathit{Sup}(r, Z \cup \{A\})$ (Lemma 10). Thus, the disjunct $Z \cup \{A\} \notin \mathit{FreqZFree}\mathcal{B}d^-(r, \sigma)$ must be true.

Because $Z \Rightarrow A$ is valid, $Z \cup \{A\}$ is a σ-frequent itemset, but not a 0-free set.

There must exist $Y \subset Z \cup \{A\}$ such that $Y \in \mathit{FreqZFree}\mathcal{B}d^-(r, \sigma)$. Y is σ-frequent (Lemma 1), and thus $Y \notin \mathit{ZFree}(r, \sigma)$. Therefore, there must exist $B \in Y$ such that $X \Rightarrow B$ is valid and that $X \subseteq Y \setminus \{B\}$. Since $Y \in \mathit{FreqZFree}\mathcal{B}d^-(r, \sigma)$, all its subsets are 0-free and thus X must be equal to $Y \setminus \{B\}$ (otherwise, $X \cup \{B\} \subset Y$ would not be a 0-free set).

Finally, the validity of $Y \setminus \{B\} \Rightarrow B$ implies that $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ (Lemmas 13 and 11). □

Lemma 16 *Let r be a binary database over a set of items R, σ be a frequency threshold, Z be a σ-frequent 0-free set and $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$. If $A \notin Z_2$ then $\forall X, X \subset Z \cup \{A\} \wedge |X| < |Z| \Rightarrow \mathit{Sup}(r, X) > \mathit{Sup}(r, Z)$.*

Proof. We prove the contrapositive.

Suppose that $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ and $\exists X, X \subset Z \cup \{A\} \wedge |X| < |Z| \wedge X \in \mathit{FreqZFree}(r, \sigma) \wedge \mathit{Sup}(r, X) \leq \mathit{Sup}(r, Z)$. Let X be such an itemset. If there are more than one such itemsets, let X be any minimal among them (minimal w.r.t. set inclusion).

$A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ implies $\mathit{Sup}(r, Z) = \mathit{Sup}(r, Z \cup \{A\})$ (Lemma 10) and further that $\mathit{Sup}(r, X) \leq \mathit{Sup}(r, Z \cup \{A\})$. By Lemma 1, $\mathit{Sup}(r, X) \geq \mathit{Sup}(r, Z \cup \{A\})$, thus $\mathit{Sup}(r, X) = \mathit{Sup}(r, Z) = \mathit{Sup}(r, Z \cup \{A\})$. By Lemma 11, $X \subset Z \cup \{A\} \wedge \mathit{Sup}(r, X) = \mathit{Sup}(r, Z \cup \{A\})$ implies that $Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)$.

By definition of X, no $X' \subset X$ satisfies $|X'| < |Z| \wedge \mathit{Sup}(r, X') \leq \mathit{Sup}(r, Z)$. Since every X', subset of X, has necessarily a smaller size than X and thus smaller than Z, the property must not hold due to the second conjunct, i.e.,

$Sup(r, X') > Sup(r, Z)$. Here $Sup(r, Z)$ is equal to $Sup(r, X)$. Since the inequality holds for every proper subset X' of X, from Lemma 14, we know that X is 0-free.

We have $X \subset Z \cup \{A\} \wedge |X| < |Z| \wedge X \in FreqZFree(r, \sigma) \wedge Sup(r, X) = Sup(r, Z) \wedge Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)$, i.e. $A \in Z_2$. □

Using these two lemmas, we finally can prove Theorem 6.

Proof. [of Theorem 6] We prove that every $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ that does not belong either to Z_1 or to Z_2, belongs to Z_3 or to Z_4.

Let $A \in \mathcal{I} \circ \mathcal{M}(r, Z) \setminus (Z_1 \cup Z_2)$.

By Lemma 15, $A \notin Z_1$ implies that $\exists Y \subset Z \cup \{A\}, \exists B \in Y, Y \setminus \{B\} \in FreqZFree(r, \sigma) \wedge B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$. Let Y be such an itemset and B be such a member of Y.

By Lemma 16, $A \notin Z_2$ implies that $\forall X \subset Z \cup \{A\}, |X| < |Z| \Rightarrow Sup(r, X) > Sup(r, Z)$, in particular $\forall X \subset (Z \cup \{A\}) \setminus \{B\}, Sup(r, X) > Sup(r, Z)$.

Three cases are possible.

The first is when $A \neq B \wedge A \notin Z$. Then $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ implies (by Lemma 10) that $Sup(r, Y \setminus \{B\}) = Sup(r, Y)$ and then (by Lemma 13) that $Gen\mathcal{M}(r, (Y \setminus \{B\}) \cup \{\overline{B}\}) = \emptyset$.

Since $Y \setminus \{B\} \subseteq (Z \cup \{A\}) \setminus \{B\}$, Lemma 2 implies that $Gen\mathcal{M}(r, ((Z \cup \{A\}) \setminus \{B\}) \cup \{\overline{B}\}) \subseteq Gen\mathcal{M}(r, (Y \setminus \{B\}) \cup \{\overline{B}\})$ $(= \emptyset)$. Therefore, no rows belong to the multiset $Gen\mathcal{M}(r, ((Z \cup \{A\}) \setminus \{B\}) \cup \{\overline{B}\})$. Thus, by Lemma 13, $Sup(r, (Z \cup \{A\}) \setminus \{B\}) = Sup(r, Z \cup \{A\}) = Sup(r, Z)$. By Lemma 14, $\forall X \subset (Z \cup \{A\}) \setminus \{B\}, Sup(r, X) > Sup(r, Z) = Sup(r, (Z \cup \{A\}) \setminus \{B\})$ implies that $(Z \cup \{A\}) \setminus \{B\}$ is a 0-free set. Therefore, A belongs to Z_4.

The second case is $A = B \wedge A \notin Z$. In this case, $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ is equivalent to $A \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{A\})$, where $(Y \setminus \{A\}) \subset (Z \cup \{A\}) \setminus \{A\}$, i.e. $Y \setminus \{A\} \subset Z$. This case turns out to be a case, where $A \in Z_3$.

The last case is when $A \in Z$. Also here, by definition of Z_3, $A \in Z_3$. □

The algorithm ConvertToFreqCl converts the condensed representations, from the DBC into the one based on frequent closed sets. In that end, ConvertToFreqCl computes the set of all frequent 0-free sets and their corresponding closures. These closures are computed in 5 steps. The first is the initialization of the closure of Z to the value of Z, followed by computing the parts $Z_1, \ldots, Z_4$ of the closure following Theorem 6.

Even though we give ConvertToFreqCl in a detailed form, the actual implementation integrates additional straightforward optimizations. Most important hints will be mentioned throughout this section.

The data structure corresponding to one itemset is called a node. A collection of such nodes is stored in an itemset-prefix-tree, like in Section 4, to allow an efficient access.

The structure of node is the following:

itemset	: set of items
support	: integer
prunedby	: set of items
closure	: array 1..4 of set of items

For a node N corresponding to an itemset Z, *N.itemset* is the set of items in Z and *N.support* is the support of Z.

N.prunedby is a set of at most two items. If it contains two different items A, B it corresponds to a set Z that is 0-free but not disjunction-free and means that $A, B \in Z$ and $Z \setminus \{A, B\} \Rightarrow A \vee B$ is valid. If *N.prunedby* contains a single item A it means that Z is not 0-free and that $Z \setminus \{A\} \Rightarrow A$ is valid. And when *N.prunedby* $= \emptyset$, the node corresponds to an itemset Z that is disjunction-free (and thus also 0-free).

The frequent itemsets are grouped in different collections according to their size and to the fact whether they are 0-free sets or not (in this latter case they are frequent itemsets of the negative border of the 0-free sets, defined in Section 7.2). For the frequent itemsets of size i, we call $\mathcal{Z}FreeFr_i$ the collection of the 0-free sets and $\mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_i$ the collection of the others. All these collections are stored in separate itemset-prefix-trees.

The four elements of the array *closure* denoted $closure_1, \ldots, closure_4$ correspond respectively to the sets $Z_1, \ldots, Z_4$ specified in Theorem 6. The following four procedures are used by the conversion algorithm to compute these four elements accordingly to the definitions of $Z_1, \ldots, Z_4$.

Algorithm 6 (update_closure_1)

Input: $\mathcal{Z}FreeFr_i$ *(fields itemset and support) nodes corresponding to all itemsets of size i from* $FreqZFree(r, \sigma)$. $\mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_{i+1}$ *(fields itemset and support) nodes corresponding to all frequent itemsets of size* $i+1$ *from* $FreqZFree\mathcal{B}d^{-}(r, \sigma)$.

Prerequisite: *$closure_1$ initialized in $\mathcal{Z}FreeFr_i$ to $\emptyset$.*

Output: *Updated $closure_1$ of all nodes from $\mathcal{Z}FreeFr_i$.*

```
for all $N \in \mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_{i+1}$ do
  for all $A \in N.itemset$ do
    Find node $N_A$ in $\mathcal{Z}FreeFr_i$ such that $N_A.itemset = N.itemset \setminus \{A\}$;
    if $N_A.support = N.support$ then
      let $N_A.closure_1 := N_A.closure_1 \cup \{A\}$;
    fi
  od
od
```

Algorithm 7 (UPDATE_CLOSURE_2)

Input: *$\mathcal{Z}FreeFr_i$ (fields itemset and support) nodes corresponding to all itemsets of size i from $FreqZFree(r, \sigma)$. $\bigcup_{j<i} \mathcal{Z}FreeFr_j$ (all fields) nodes corresponding to all itemsets of sizes strictly lower than i from $FreqZFree(r, \sigma)$.*

Prerequisite: *$closure_2$ initialized in $\mathcal{Z}FreeFr_i$ to $\emptyset$.*

Output: *Updated $closure_2$ of all nodes from $\mathcal{Z}FreeFr_i$.*

```
for all $N_C \in \bigcup_{j<i} \mathcal{Z}FreeFr_j$ do
  for all $N \in \mathcal{Z}FreeFr_i$ such that $N.itemset \subset \bigcup_{k=1..4} N_C.closure_k$
   and $N_C.support = N.support$ do
    let $N.closure_2 := N.closure_2 \cup \bigcup_{k=1..4} N_C.closure_k$;
  od
od
```

Algorithm 8 (UPDATE_CLOSURE_3)

Input: *$\mathcal{Z}FreeFr_i$ (fields itemset and support) nodes corresponding to all itemsets of size i from $FreqZFree(r, \sigma)$. $\mathcal{Z}FreeFr_{i-1}$ (all fields) nodes corresponding to all itemsets of size $i - 1$ from $FreqZFree(r, \sigma)$.*

Prerequisite: *$closure_3$ of each node N in $\mathcal{Z}FreeFr_i$ initialized to N.itemset.*

Output: *Updated $closure_3$ of all nodes from $\mathcal{Z}FreeFr_i$.*

```
for all $N \in \mathcal{Z}FreeFr_i$ do
  for all $A \in N.itemset$ do
    Find node $N_A$ in $\mathcal{Z}FreeFr_{i-1}$ such that $N_A.itemset = N.itemset \setminus \{A\}$;
    let $N.closure_3 := N.closure_3 \cup \bigcup_{k=1..4} N_A.closure_k$;
  od
od
```

Algorithm 9 (UPDATE_CLOSURE_4)

Input: *$\mathcal{Z}FreeFr_i$ (fields itemset, $closure_3$ and support) nodes corresponding to all itemsets of size i from $FreqZFree(r, \sigma)$.*

Prerequisite: *$closure_4$ initialized in $\mathcal{Z}FreeFr_i$ to $\emptyset$.*

Output: *Updated $closure_4$ of all nodes from $\mathcal{Z}FreeFr_i$.*

```
for all $N_W \in \mathcal{Z}FreeFr_i$ do
  for all $B \in N_W.closure_3 \setminus N_W.itemset$ do
      for all $A \in N_W.itemset$ do // here $A \neq B$
          Find node $N_Z$ in $\mathcal{Z}FreeFr_i$ such that
            $N_Z.itemset = (N_W.itemset \cup \{B\}) \setminus \{A\}$;
          if $N_Z.support = N_W.support$ then
              let $N_Z.closure_4 := N_Z.closure_4 \cup \{A\}$;
          fi
      od
  od
od
```

Implementing these procedures using a single field *closure* instead of an array of 4 elements $closure_1, \ldots, closure_4$ is described in details in [20]. This improvement avoids merging the items each time a complete closure is needed, e.g., in algorithms UPDATE_CLOSURE_2 (lines 2 and 3) and in UPDATE_CLOSURE_3 (line 4).

The conversion algorithm uses the procedure FINDDISJRULE described in Section 4 to compute *prunedby* for itemsets that are 0-free but not disjunction-free and also the following procedure FINDSIMPRULE to determine *prunedby* for non-0-free sets.

Algorithm 10 (FINDSIMPRULE)

Input: *Itemset Z, S support of Z, collection of nodes $\mathcal{C}$ including nodes corresponding to all proper subsets of Z of size $|Z| - 1$.*

Output: *Itemset P containing 1 item corresponding to the right-hand-side item of a valid rule $Z \setminus \{A\} \Rightarrow A$ if it exists, $\emptyset$ otherwise.*

```
Find any item $A$ in $Z$ and any node $N_A$ in $\mathcal{C}$ such that
      $N_A.itemset = Z \setminus \{A\}$ and $S = N_A.support$;
if such $A$ exists then // in this case $Z \setminus \{A\} \Rightarrow A$ is valid
      let $P := \{A\}$;
else
      let $P := \emptyset$;
fi
output $P$;
```

Now, we give the conversion algorithm itself.

Algorithm 11 (ConvertToFreqCl)

Input: σ, *$DBC(r,\sigma)$ of the form $\langle \mathcal{F}, \mathcal{B} \rangle$ and for each itemset $X \in \mathcal{F} \cup \mathcal{B}$ its support $Sup(r, X)$.*

Prerequisite: *$\mathcal{F} \cup \mathcal{B}$ not empty.*

Output: *$FreqClosed(r, \sigma)$ and their supports.*

```
for all $X \in \mathcal{F}$ do // Disjunction-free input sets
    Create new node $N_X$ in $\mathcal{Z}FreeFr_{|X|}$ with $N_X.support := Sup(r, X)$,
      $N_X.itemset := X$, $N.closure := [\emptyset, \emptyset, X, \emptyset]$, $N_X.prunedby := \emptyset$;
od
for all $X \in \mathcal{B}$ do // Non-disjunction-free input sets
    let $P := FindSimpRule(X, Sup(r, X), \mathcal{Z}FreeFr_{|X|-1})$;
    if $P = \emptyset$ then
        Create new node $N_X$ in $\mathcal{Z}FreeFr_{|X|}$ with $N_X.support := Sup(r, X)$,
          $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, X, \emptyset]$,
          $N_X.prunedby := FindDisjRule(X, Sup(r, X), \mathcal{Z}FreeFr_{|X|-1} \cup \mathcal{Z}FreeFr_{|X|-2})$;
    else
        Create new node $N_X$ in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_{|X|}$ with $N_X.support := Sup(r, X)$,
          $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, \emptyset, \emptyset]$, $N_X.prunedby := P$;
    fi
od

Find node $N_E$ in $\mathcal{Z}FreeFr_0$ such that $N_E.itemset = \emptyset$;
let $N_E.closure_1 := \{N_Y.itemset | N_Y \in \mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_1\}$;
let $i := 1$;
while $\mathcal{Z}FreeFr_{i-1} \neq \emptyset$ do
    let $\mathcal{C}_{i+1} := APRIORIGen(\mathcal{Z}FreeFr_i)$;
    for all $X \in \mathcal{C}_{i+1}$ do
        Find node $N$ in $\mathcal{Z}FreeFr_i$ such that $N.itemset \subset X$ and $|N.prunedby| = 2$;
        if such $N$ exists then
            let $A, B$ be items such that $N.prunedby = \{A, B\}$;
            Find nodes $N_A, N_B$ in $\mathcal{Z}FreeFr_i$ and $N_{AB}$ in $\mathcal{Z}FreeFr_{i-1}$ such that
              $N_A.itemset = X \setminus \{A\}$ and $N_B.itemset = X \setminus \{B\}$ and
              $N_{AB}.itemset = X \setminus \{A, B\}$;
            let $S := N_A.support + N_B.support - N_{AB}.support$;
            if $S \geq \sigma$ then
                let $P := FindSimpRule(X, S, \mathcal{Z}FreeFr_i)$;
                if $P = \emptyset$ then
                    Create new node $N_X$ in $\mathcal{Z}FreeFr_{|X|}$ with $N_X.support := S$,
                      $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, X, \emptyset]$, $N_X.prunedby := \{A, B\}$;
                else
                    Create new node $N_X$ in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_{|X|}$ with $N_X.support := S$,
                      $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, \emptyset, \emptyset]$, $N_X.prunedby := P$;
                fi
            fi
        fi
```

```
32.     od
33.     UPDATE_CLOSURE_1 ($\mathcal{Z}FreeFr_i, \mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{i+1}$);
34.     UPDATE_CLOSURE_2 ($\mathcal{Z}FreeFr_i, \bigcup_{j<i} \mathcal{Z}FreeFr_j$);
35.     UPDATE_CLOSURE_3 ($\mathcal{Z}FreeFr_i, \mathcal{Z}FreeFr_{i-1}$);
36.     UPDATE_CLOSURE_4 ($\mathcal{Z}FreeFr_i$);
37.     let $i := i + 1$;
38. od
39. output $\{\langle \bigcup_{k=1..4} N.closure_k, N.support\rangle | N \in \bigcup_{j<i} \mathcal{Z}FreeFr_j\}$;
```

In lines 1-11, Algorithm ConvertToFreqCl loads the input and converts it to a collection of nodes, one node per itemset. For an itemset X, if the call to FindSimpRule gives a nonempty result (in line 5), then X is not 0-free, but it is an element of the set $FreqZFree\mathcal{B}d^{-}(r, \sigma)$ (frequent itemsets of the negative border of 0-free sets). Thus, it is stored in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{|X|}$ and has a field *prunedby* filled with a single item. The nodes corresponding to frequent 0-free sets of $\mathcal{B}d^{-}(r, \sigma)$ (the negative border of disjunction-free sets) are distinguished when loaded in line 7 by the content of the field *prunedby*, which contains two items.

Reading the input is followed by the computation of the closure of $\emptyset$ in lines 12 and 13, according to Theorem 6 applied to this particular itemset. All other frequent closed sets are obtained by computing the closures of nonempty frequent 0-free sets. This leads to two interleaved processes. The first is the creation of nodes corresponding to frequent 0-free sets or to frequent itemsets from the negative border of 0-free sets. And the second is the construction of the frequent closed sets by computing the closures of frequent 0-free sets.

These two processes are performed within the loop starting at line 15. The i^{th} iteration first generates the itemsets of size $i+1$ that are either frequent 0-free sets (stored in $\mathcal{Z}FreeFr_{i+1}$, line 26) or frequent itemsets from the negative border of 0-free sets (stored in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{i+1}$, line 28). Then it determines the closures of the 0-free sets of size i using the characterization of the closed sets stated in Theorem 6. It should be noticed that during this i^{th} iteration the algorithm cannot compute the closures of the newly generated 0-free sets of size $i+1$ since the use of Theorem 6 requires in this case the frequent itemsets from the negative border of 0-free sets of size $i+2$ (i.e., $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{i+2}$).

In lines 16-19, the algorithm selects candidate sets that are potentially frequent and 0-free, but are not in the input because they are proper supersets of non-disjunction-free sets. Next, in lines 20-22, for each candidate set X, the algorithm searches the necessary information to restore the support of X. It takes Y, any non-disjunction-free proper subset of X, and considers A, B the items of *N.prunedby* in the node N corresponding to Y. Since $Y \setminus \{A, B\} \Rightarrow A \vee B$ is a valid simple disjunctive rule then, using Lemma 5, the algorithm infers the support S of X.

If X is frequent (line 23) then the corresponding node N_X is created in lines 24-29. If the call to $FindSimpRule$ on X returns an empty set then X is a 0-free set, threfore N_X is created in $\mathcal{Z}FreeFr_{i+1}$ and the value of *pruneby* for Y propagates to X. If not, the set X is not a 0-free set, but all its subsets are 0-free sets and thus X is in the negative border of 0-free sets. In this case N_X is created in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}Bd_{i+1}$, and N_X.*pruneby* is filled with the result of $FindSimpRule$.

The computation of the closures of the 0-free sets of size i (stored in $\mathcal{Z}FreeFr_i$) is then performed in lines 33-36.

The algorithm exits the main loop when the closures of frequent 0-free sets of the maximal size are computed.

Theorem 7 *The algorithm* ConvertToFreqCl *regenerates all and only σ-frequent 0-free sets in $\bigcup_{j<k} \mathcal{Z}FreeFr_j$, where k is the value of the variable i at the end of the execution of* ConvertToFreqCl.

Proof. Similar to the proof of Theorem 3. □

Lemma 17 *Given r a binary database over R, σ a support threshold and X a σ-frequent closed set w.r.t. r, $\exists Y \subseteq X, Y \in FreqZFree(r,\sigma)$ and $X = \mathcal{I} \circ \mathcal{M}(r,Y)$.*

Proof. Let r, σ and X be same as in the lemma. Let $Y \subseteq X$ be the smallest itemset w.r.t. itemset inclusion having the same support as the support of X. Below, we show that Y is σ-frequent 0-free set and that $X = \mathcal{I} \circ \mathcal{M}(r,Y)$.

By Lemma 11, $Sup(r,X) = Sup(r,Y)$ implies $X \subseteq \mathcal{I} \circ \mathcal{M}(r,Y)$. Since $X = \mathcal{I} \circ \mathcal{M}(r,X)$ (X is closed), $\mathcal{I} \circ \mathcal{M}(r,X) \subseteq \mathcal{I} \circ \mathcal{M}(r,Y)$. By Lemma 9, $\mathcal{I} \circ \mathcal{M}(r,X) \supseteq \mathcal{I} \circ \mathcal{M}(r,Y)$. Therefore, $X = \mathcal{I} \circ \mathcal{M}(r,X) = \mathcal{I} \circ \mathcal{M}(r,Y)$.

Since Y is the smallest itemset having the same support, $\forall Z \subset Y, Sup(r,Z) \neq Sup(r,Y)$. Additionally, by the anti-monotonicity of support, we can refine that property into $\forall Z \subset Y, Sup(r,Z) > Sup(r,Y)$, which according to Lemma 14, implies that Y is 0-free set. $Sup(r,X) = Sup(r,Y)$ and X σ-frequent implies that Y is σ-frequent. □

Theorem 8 (Correctness of ConvertToFreqCl) *The algorithm* ConvertToFreqCl *outputs all and only σ-frequent closed sets along with their supports.*

Proof. Immediate from Theorem 7 and Lemma 17. □

We have presented the advantage of the DBC w.r.t. closed sets in terms of representation size in Section 5. In this section, we report experiments showing that the frequent closed sets can be derived very efficiently from the DBC representation using the algorithm ConvertToFreqCl. These additional experiments lead to the conclusion that in practice the DBC turns out to be also an interesting condensed representation of the frequent closed sets.

These experiments are run in the same conditions as in Section 5 (same platform and data sets, and same implementations of CLOSE [12] and CLOSET [21]). We compare the extraction times of direct computation of frequent closed sets using CLOSE and CLOSET with the extraction of the DBC (by means of HLinEx and VLinEx) followed by its conversion into frequent closed sets using an implementation of ConvertToFreqCl. The extraction times of the DBC alone (without running ConvertToFreqCl) have been given in Section 6.3. The running times (in seconds) for several support thresholds are given in Figure 5 (note that some axes are logarithmically scaled).

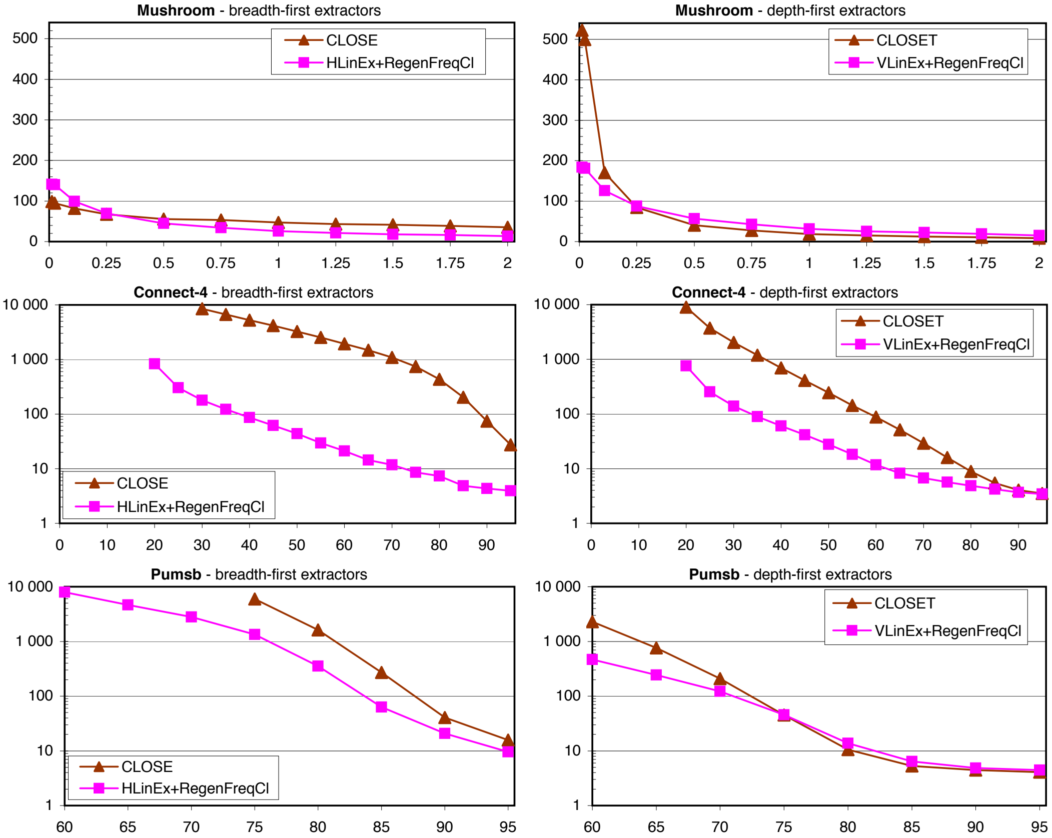


Fig. 5. Experiments on the three data sets with breadth-first (left) and depth-first (right) algorithms, reporting the extraction time in seconds vs the relative support threshold in %.

The results on the *Mushroom* data set are the most ambiguous. All resulting times are very close, and for both strategies the curves related to the extraction of DBC followed by ConvertToFreqCl intersect the ones of the corresponding reference algorithms (see Figure 5, upper graphics). On the *Connect-4* data set, the extraction of the DBC followed by its conversion into closed sets is always faster (up to 15 times with depth-first extractors and up to 100 times with breadth-first ones). For the last data set, *Pumsb*, mining frequent closed sets was difficult for all techniques and has been stopped at a relative support threshold of 60%. In the most difficult cases (low support thresholds), the experiments show a real gain, up to fivefold speed-up, when using the DBC and its conversion versus a direct extraction of the closed sets.

In most of experiments, the extraction of DBC followed by ConvertToFreqCl produces the frequent closed sets much more efficiently than the direct extraction of the frequent closed sets. In the cases where the direct extraction is still faster, it should be noticed that the performances of both approaches are very close.

In [20], we experimentally demonstrate that the regeneration of the frequent closed sets from the DBC using ConvertToFreqCl is quasi-linear w.r.t. the flat-storage size of frequent closed set collection (i.e., quasi-linear w.r.t. ConvertToFreqCl output size).

8 Conclusion

Knowledge discovery tasks based on frequent itemset extraction are generally very difficult at low support thresholds because the direct extraction of the frequent itemsets turns out to be a very long process. In this paper, we proposed a condensed representation of frequent itemsets, called disjunction-bordered condensation, that can be extracted more efficiently and that can be used to regenerate all frequent itemsets and their exact supports.

We proposed two algorithms to extract this representation, based respectively on a depth-first and on a breadth-first strategy. We presented experiments showing that the size of the DBC is very small w.r.t. the size of the collection of all frequent itemsets, and that the DBC can be extracted much more efficiently than frequent itemsets.

We compared the DBC with the representation based on frequent closed sets [12], which is the other condensed representation investigated in the literature allowing an exact regeneration of the frequencies. We showed that in general the DBC representation is smaller and can be extracted much more efficiently than the frequent closed sets. In the cases where the extraction of the DBC is not faster or where the DBC representation is not smaller, the experiments report very close measures.

We also presented a procedure to generate the frequent closed sets from the DBC representation. The experiments show that this conversion can be done very efficiently, and that in most cases it is faster to extract the DBC and then to derive the frequent closed sets than to extract them directly. Here again, in the cases where the approach based on the DBC representation is not faster the performances remain very similar. Thus the DBC representation turns out to be an interesting condensed representation for the frequent closed sets themselves.

Future work. Frequent itemset mining is often followed by some data mining tasks making use of frequent itemsets, and it is a common practice to use some well-defined subcollections of frequent itemsets and to discard remaining ones. In such cases, the results of this paper could be refined to regenerate such subcollections, circumventing often voluminous collections of all frequent itemsets. For example, the regeneration procedure of all frequent itemsets can be modified to extract frequent itemsets under constraints [23,24]. Also, the algorithm converting the DBC into frequent closed sets can be adapted to extract frequent δ-free sets [13,14].

This paper is limited to the use of simple disjunctive rules of the form $Y \Rightarrow A \vee B$ to condense the representation of the frequent itemsets. An interesting direction for future work is to consider the condensation based on rules involving 2-disjunction rules (e.g. $Y \Rightarrow A \vee B \vee C$) or even n-disjunction rules for arbitrary n. An open question is then, what will be the overall gain of the approach ? Will the speed-up stemming from a more important pruning compensate the overhead due to the handling of these rules ?

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proc. of the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago de Chile, Chile, 1994, pp. 487 – 499.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo, Fast discovery of association rules, in: Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 307–328.

[3] H. Toivonen, Sampling large databases for association rules, in: Proc. of the 22th International Conference on Very Large Data Bases (VLDB'96), Bombay, India, 1996, pp. 134 – 145.

[4] S. Brin, R. Motwani, J. D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, in: Proc. of the ACM SIGMOD

International Conference on Management of Data (SIGMOD'97), Tucson, Arizona, USA, 1997, pp. 255–264.

[5] R. C. Agarwal, C. C. Aggarwal, V. V. Prasad, A tree projection algorithm for finding frequent itemsets, Journal of Parallel and Distributed Computing 61 (3) (2001) 350–371.

[6] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proc. of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, Texas, 2000, pp. 1–12.

[7] L. Dehaspe, L. D. Raedt, Mining association rules in multiple relations, in: S. Džeroski, N. Lavrač (Eds.), Proc. of the 7th International Workshop on Inductive Logic Programming (ILP'97), Vol. 1297 of LNCS, Springer-Verlag, Prague, Czech Republic, 1997, pp. 125–132.

[8] V. Crestana-Jensen, N. Soparkar, Frequent itemset counting across multiple tables, in: Proc. of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00), Vol. 1805 of LNAI, Springer-Verlag, Kyoto, Japan, 2000, pp. 49–61.

[9] B. Liu, W. Hsu, Y. Ma, Integrating classification and association rule mining, in: Proc. of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98), New York City, New York, USA, 1998, pp. 80–86.

[10] J. Dougherty, R. Kohavi, M. Sahami, Supervised and unsupervised discretization of continuous features, in: Proc. of the 12th International Conference on Machine Learning (ICML'95), Tahoe City, California, USA, 1995, pp. 194–202.

[11] U. M. Fayyad, K. B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proc. of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry, France, 1993, pp. 1022–1029.

[12] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Efficient mining of association rules using closed itemset lattices, Information Systems 24 (1) (1999) 25–46.

[13] J.-F. Boulicaut, A. Bykowski, C. Rigotti, Approximation of frequency queries by mean of free-sets, in: Proc. of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00), Lyon, France, 2000, pp. 75–85.

[14] J.-F. Boulicaut, A. Bykowski, C. Rigotti, Free-sets : a condensed representation of boolean data for the approximation of frequency queries, Data Mining and Knowledge Discovery 7 (1) (2003) 5–22.

[15] H. Mannila, H. Toivonen, Multiple uses of frequent sets and condensed representations, in: Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), Portland, USA, 1996, pp. 189–194.

[16] Y. Bastide, N. Pasquier, R. Taouil, L. Lakhal, G. Stumme, Mining minimal non-redundant association rules using frequent closed itemsets, in: Proc. of the 6th International Conference on Rules and Objects in Databases (DOOD'00), Vol. 1861 of LNCS, Springer-Verlag, London, UK, 2000, pp. 972–986.

[17] A. Bykowski, C. Rigotti, A condensed representation to find frequent patterns, in: Proc. of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'01), Santa Barbara, CA, USA, 2001, pp. 267–273.

[18] H. Mannila, H. Toivonen, Levelwise search and borders of theories in knowledge discovery, Data Mining and Knowledge Discovery 1 (3) (1997) 241–258.

[19] R. J. Bayardo, Efficiently mining long patterns from databases, in: Proc. of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98), Seattle, Washington, USA, 1998, pp. 85–93.

[20] A. Bykowski, Condensed representations of frequent sets : Application to descriptive pattern discovery, Ph.D. thesis, INSA-Lyon, 20, Avenue A. Einstein, F-69621 Villeurbanne Cedex, France (Oct. 2002).

[21] J. Pei, J. Han, R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, in: Proc. of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'00), Dallas, Texas, 2000, pp. 21–30.

[22] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proc. of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93), Washington, D.C., USA, 1993, pp. 207–216.

[23] R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in: D. Heckerman, H. Mannila, D. Pregibon, R. Uthurusamy (Eds.), Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining, (KDD'97), Newport Beach, California, USA, 1997, pp. 67–73.

[24] B. Jeudy, J.-F. Boulicaut, Optimization of association rule mining queries, Intelligent Data Analysis 6 (4) (2002) 341–357.