Avoiding the itemset closure computation "pitfall"

Tarek Hamrouni, Sadok Ben Yahia and Yahya Slimani

Faculté des Sciences de Tunis
Département des Sciences de l'Informatique,
Campus Universitaire, 1060 Tunis, Tunisie.
{sadok.benyahia, yahya.slimani}@fst.rnu.tn

Abstract. Extracting generic bases of association rules seems to be a promising issue in order to present informative and compact user added-value knowledge. However, extracting generic bases requires partially ordering costly computed itemset closures. To avoid the nightmarish itemset closure computation cost, specially for sparse contexts, we introduce an algorithm, called Prince, allowing an astute extraction of generic bases of association rules. The Prince algorithm main originality is that the partial order is maintained between frequent minimal generators and no more between frequent closed itemsets. A structure called *minimal generator lattice* is then built, from which the derivation of itemset closures and generic association rules becomes straightforward. An intensive experimental evaluation, carried out on benchmarking and "worst case" datasets, showed that Prince largely outperforms the pioneer algorithms, *i.e.*, Close, A-Close and Titanic.
Keywords: Data mining, Formal Concept Analysis, generic association rule bases, *minimal generator lattice*.

1 Introduction

It is widely known that frequent itemset based algorithms suffer from the generation of a very large number of frequent itemsets and hence association rules. Thus, this prohibitive generation reduces not only efficiency but also effectiveness of the mined knowledge. In fact, users have to perform tedious rummage within an overwhelming large number of mined association rules [1]. In this context, the approach based on the extraction of frequent *closed* itemsets [2] presented a clear promise to reduce the frequent itemset extraction cost and mainly to offer, to users, irreducible nuclei of association rules that are commonly known as "generic bases" of association rules. This approach, relying on the Formal Concept Analysis mathematical background [3], proposes to reduce the search space by detecting intrinsic structural properties. Therefore, the problem of mining association rules might be reformulated, under the (frequent) closed itemsets discovery point of view, as follows [4]:

1. Discover both distinct "closure systems", *i.e.*, sets of sets which are closed under the intersection operator, namely the set of closed itemsets and the set

Radim Bělohlávek, Václav Snášel (Eds.): CLA 2005, pp. 46–59, ISBN 80–248–0863–3.

of minimal generators. Also, the *upper cover* (Cov^u) of each closed itemset should be available.
2. From all discovered information during the first step, *i.e.*, both closure systems and the upper cover sets, derive generic bases of association rules (from which all remaining association rules can be derived).

The essential report after an overview of the state of the art of frequent closed itemset based algorithms (*e.g.*, [1, 2, 5–9]) can be summarized in what follows:

1. These algorithms mainly concentrate on the first task, *i.e.*, reducing the computation time of the frequent itemset extraction step. Their performances are interesting on dense contexts. However, they present modest performances on sparse contexts. Indeed, computing itemset closures in this type of contexts is heavily handicapping on these algorithm performances, since frequent closed itemset search space tends to overlap that of frequent itemsets.
2. The frequent closed itemset based algorithms neglect the second task, *i.e.*, extracting generic association rule bases. Indeed, none of them accepted to maintain the order covering the relationship between frequent closed itemsets.

In this paper, we propose a new algorithm, called PRINCE, aiming to extract generic bases of association rules. PRINCE performs a level-wise browsing of the search space. Its main originality is that it is the only one who accepted to bear the cost of building the partial order. Interestingly enough, to amortize this prohibitive cost, the partial order is maintained between frequent minimal generators and no more between frequent closed itemsets. The obtained partially ordered structure is called *minimal generator lattice* [10], in which each equivalence class is reduced to the corresponding set of frequent minimal generators. Hence, itemset closures are not computed but derived when PRINCE performs a simple sweeping of the *minimal generator lattice* to derive generic bases of association rules. Practical performances of the PRINCE algorithm have been compared to those of well known level-wise browsing algorithms, *i.e.*, CLOSE [2], A-CLOSE [5], and TITANIC [6]. Our experiments were carried out on benchmark datasets (dense and sparse) and on "worst case" datasets. Obtained results are very encouraging: although our algorithm performs the partial order construction task, it largely outperforms CLOSE, A-CLOSE, and TITANIC algorithms. In addition to the "worst case" datasets and due to space limit, we report our results only on two benchmark datasets, frequently used for evaluating data mining algorithms.

It is important to note that omitting to compare PRINCE performances to those of more recent algorithms, *e.g.*, LCM [8], DCI-CLOSED [9], is argued by two reasons:

1. CLOSE, A-CLOSE and TITANIC algorithms determine at least the "key" information provided by the frequent minimal generator set.
2. Following our claim that stressing on fast enumeration of frequent closed itemsets will not be of any interest nor presents any added-value knowledge for end-users, since not all required information is extracted.

The remainder of the paper is organized as follows. Section 2 sketches the generic association rule basis extraction problem. Section 3 is dedicated to the presentation of PRINCE algorithm. Experimental results showing the utility of the proposed approach are reported in section 4. The conclusion and future work are presented in section 5.

2 Generic association rule basis extraction

Since the apparition of the approach based on the extraction of frequent closed itemsets [2], several generic association rule bases were introduced among which those of Bastide *et al.* [11] and which are defined as follows:

1. The *generic basis for exact association rules* is defined as follows:

   Definition 1. *Let $\mathcal{FCI}_{\mathcal{K}}$ be the set of frequent closed itemsets extracted from the extraction context $\mathcal{K}$. For each entry f in $\mathcal{FCI}_{\mathcal{K}}$, let MG_f be the set of its minimal generators. The generic basis for exact association rules GB is given by: $GB = \{R\text{: } g \Rightarrow (f - g) \mid f \in \mathcal{FCI}_{\mathcal{K}}$ and $g \in MG_f$ and $g \neq f$[1]$\}$.*

2. The transitive reduction of the informative basis [11], which is a basis for all approximate association rules, is defined as follows[2]:

   Definition 2. *Let $\mathcal{FMG}_{\mathcal{K}}$ be the set of frequent minimal generators extracted from the extraction context $\mathcal{K}$. The transitive reduction RI is given by: $RI = \{R \mid R\text{: } g \Rightarrow (f - g) \mid f \in \mathcal{FCI}_{\mathcal{K}}$ and $g \in \mathcal{FMG}_{\mathcal{K}}$ and $g'' \prec f$[3] and $Conf(R) \geq minconf\}$.*

In the remainder of the paper, we will refer to the generic association rules formed by the couple (GB, RI). This couple is informative, sound and lossless [11, 12] and the association rules forming it are referred as *informative association rules*. Thus, given an *Iceberg Galois lattice* – in which each frequent closed itemset is decorated by its list of minimal generators – the derivation of these association rules can be performed straightforwardly. Indeed, approximate generic association rules represent "inter-node" implications, assorted with the confidence measure, between two adjacent comparable equivalence classes, *i.e.*, from a frequent closed itemset to another frequent closed itemset immediately covering it. For example, referring to the *Iceberg Galois lattice* depicted by Figure 1 (Right), the approximate generic association rule $C \overset{0.75}{\Rightarrow} ADEF$ is generated from both equivalence classes topped respectively by the frequent closed itemsets "CE" and "ACDEF". Inversely, exact generic association rules are "intra-node" implications, with a confidence equal to 1, extracted from each node in the partially ordered structure. For example, from the closed itemset "ACDEFG", three exact generic association rules are obtained: AG⇒CDEF, DG⇒ACEF and FG⇒ACDE.

[1] The condition $g \neq f$ ensures discarding non-informative association rules of the form $g \Rightarrow \emptyset$.
[2] The closure operator is noted $''$.
[3] The notation $\prec$ indicates that f covers g'' in the *Iceberg Galois lattice*.

	A	B	C	D	E	F	G
1	×		×	×	×	×	×
2	×	×	×	×	×	×	
3		×		×	×		
4		×	×		×		×
5	×		×	×	×	×	×

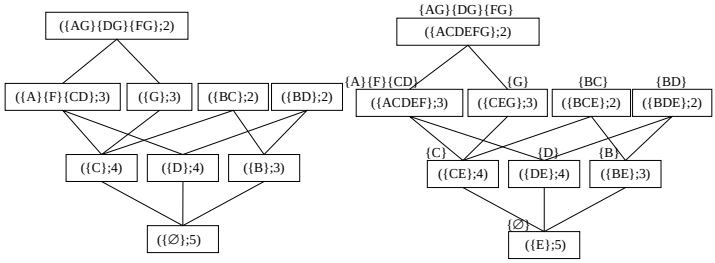

Fig. 1. Left: Extraction context $\mathcal{K}$. **Center**: The associated *minimal generator lattice* for *minsup*=2. **Right**: The associated *Iceberg Galois lattice* for *minsup*=2.

3 Prince algorithm

In order to palliate the frequent closed itemset based algorithm insufficiencies, *i.e.*, the cost of the closure computation as well as neglecting the partial order construction, we will introduce a new algorithm called Prince. Prince highly reduces the cost of closure computation and generates the partially ordered structure, which makes it able to extract straightforwardly generic association rule bases without coupling it with another algorithm. Prince takes as input an extraction context $\mathcal{K}$ where the items are sorted by lexicographic order, the minimum threshold of support *minsup* and the minimum threshold of confidence *minconf*. It outputs the list of frequent closed itemsets and their associated minimal generators as well as the informative association rules formed by the couple (GB, RI). Thus, Prince operates in three successive steps: (i) Minimal generator determination (ii) Partial order construction (iii) Generic association rule basis extraction.

3.1 Minimal generator determination

Following the "Test-and-generate" technique, Prince traverses the search space by level to determine the set of frequent minimal generators $\mathcal{FMG}_{\mathcal{K}}$ sorted by decreasing support values. $\mathcal{FMG}_{\mathcal{K}}$ is then considered as divided into several subsets. Each subset represents a given support. Thus, each time that a frequent minimal generator is determined, it is added to the subset representing its support. Prince also keeps track of the negative border of minimal generators $\mathcal{GB}\mathrm{d}^{-}$ [4] [13]. In the second step, the set of frequent minimal generators will serve as a backbone to construct the *minimal generator lattice*. As shown by the following property, the union of $\mathcal{FMG}_{\mathcal{K}}$ and $\mathcal{GB}\mathrm{d}^{-}$ will be used, in the second step, as a concise lossless representation of frequent itemsets:

Property 1. [13] Let X be an itemset. If $\exists\ Z \in \mathcal{GB}\mathrm{d}^{-}$ and $Z \subseteq X$ then X is infrequent. Otherwise, X is frequent and $\mathrm{Supp}(X) = \min\{\mathrm{Supp}(g) \mid g \in \mathcal{FMG}_{\mathcal{K}}$ and $g \subseteq X\}$.

[4] An itemset belong to $\mathcal{GB}\mathrm{d}^{-}$ if it is an infrequent minimal generator and all its subsets are frequent minimal generators.

PRINCE uses, in this step, the same pruning strategies introduced in TITANIC namely *minsup*, the ideal order of the frequent minimal generator set and the estimated support. A *trie* is used to store the minimal generator set in order to speed-up the extraction of information that will be later of use. The path from the root to each node represents a minimal generator.

3.2 Partial order construction

In this step, the frequent minimal generator set $\mathcal{FMG}_{\mathcal{K}}$ will form a *minimal generator lattice*, and this *without any access to the extraction context*. The main idea is how to construct the partial order without computing itemset closures, *i.e.*, how guessing the subsumption relation by only comparing minimal generators? To achieve this goal, the list of immediate successors[5] of each equivalence class will be updated in an iterative way. The processing of the frequent minimal generator set is done according to the order imposed in the first step (*i.e.*, by decreasing support values). Each frequent minimal generator g of size k ($k \geq 1$) is introduced into the minimal generator lattice by comparing it to the immediate successors of its (k-1)-subsets[6]. This is based on the *isotony* property of the closure operator [14]. Indeed, let g_1, a (k-1)-itemset, be one of the subsets of g, $g_1 \subset g \Rightarrow g_1'' \subset g''$. Thus, the equivalence class to which belongs g is a successor (not necessarily an immediate one) of the equivalence class to which belongs g_1.

While comparing g to the immediate successor list of g_1, noted L, two cases are to be distinguished. If L is empty then g is added to L. Otherwise, g is compared to the elements already belonging to L (*cf.* Proposition 1). The imposed order in the first step allows to distinguish only two cases sketched by Proposition 1 by replacing the frequent minimal generators X and Y by respectively g and one of the elements of L.

Proposition 1. *[15] Let X, $Y \in \mathcal{FMG}_{\mathcal{K}}$, $\mathcal{C}_X$ and $\mathcal{C}_Y$ their respective equivalence classes:*
a. If Supp(X) = Supp(Y) = Supp($X \cup Y$) then X and Y belong to the same equivalence class.
b. If Supp(X) < Supp(Y) and Supp(X) = Supp($X \cup Y$) then $\mathcal{C}_X$ (resp. $\mathcal{C}_Y$) is a successor (resp. predecessor) of $\mathcal{C}_Y$ (resp. $\mathcal{C}_X$).

The computation of the support of $(X \cup Y)$ is performed in a direct manner if $(X \cup Y)$ belongs to $\mathcal{FMG}_{\mathcal{K}} \cup \mathcal{GB}\mathrm{d}^{-}$. $\mathcal{C}_X$ and $\mathcal{C}_Y$ are then incomparable. Otherwise, Property 1 is applied. The support computation stops then as soon as we find a minimal generator that is included in $(X \cup Y)$ and has a support strictly lower than that of X and that of Y. $\mathcal{C}_X$ and $\mathcal{C}_Y$ are then incomparable.

During these comparisons and to avoid redundant closure computations, PRINCE introduces two complementary functions. These functions make it possible to maintain the concept of equivalence class throughout processing. To this

[5] By the term "immediate successor", we indicate a frequent minimal generator, unless otherwise specified.

[6] In the first step and for each k-candidate, links towards its (k-1)-subsets are stored during the check of the ideal order property.

end, each equivalence class $\mathcal{C}$ will be characterized by a *representative* itemset, which is the *first* frequent minimal generator introduced into the *minimal generator lattice*. Both functions are described below:

1. MANAGE-EQUIV-CLASS: This function is used if a frequent minimal generator, say g, is compared to the representative itemset of its equivalence class, say $\mathcal{R}$. The MANAGE-EQUIV-CLASS function replaces all occurrences of g by $\mathcal{R}$ in the immediate successor lists in which g was added. Then, comparisons to carry out with g will be made with $\mathcal{R}$. Thus, for each equivalence class, only its representative itemset appears in the lists of immediate successors.

2. REPRESENTATIVE: This function makes it possible to find, for each frequent minimal generator g, the representative $\mathcal{R}$ of its equivalence class in order to complete the immediate successor list of $\mathcal{C}_g$. This allows to manage only one immediate successor list for all frequent minimal generators belonging to the same equivalence class.

The pseudo-code of the second step is given by the GEN-ORDER procedure (Algorithm 1). Each entry, say g, in $\mathcal{FMG}_\mathcal{K}$ is composed by the following fields: (i) `support`: the support of g (ii) `direct-subsets`: the list of (k-1)-subsets of g (iii) `immediate-succs`: the list of immediate successors of g. At the end of the execution of the GEN-ORDER procedure, g.`immediate-succs` is empty if g is not the representative itemset of its equivalence class or if g belongs to a maximal equivalence class, *i.e.*, not subsumed by any equivalence class. Otherwise, this list will contain only representative frequent minimal generators.

Algorithm 1 GEN-ORDER

Require: - $\mathcal{FMG}_\mathcal{K}$.
Ensure: - The elements of $\mathcal{FMG}_\mathcal{K}$ partially ordered in the form of a *minimal generator lattice*.

for all ($g \in \mathcal{FMG}_\mathcal{K}$) **do**
  for all ($g_1 \in g$.`direct-subsets`) **do**
    $\mathcal{R}$ = REPRESENTATIVE(g_1);
    for all ($g_2 \in \mathcal{R}$.`immediate-succs`) **do**
      if (g.`support` = g_2.`support` = Supp($g \cup g_2$)) **then**
        MANAGE-EQUIV-CLASS(g,g_2); /*g, $g_2 \in \mathcal{C}_g$ and g_2 is the representative of $\mathcal{C}_g$*/
      else if (g.`support` < g_2.`support` and g.`support` = Supp($g \cup g_2$)) **then**
        g is compared with g_2.`immediate-succs`;
        /*For the remainder of the element of $\mathcal{R}$.`immediate-succs`, g is compared only with each $g_3 \mid g_3$.`support` > g.`support`;*/
      end if
    end for
    if ($\forall\ g_2 \in \mathcal{R}$.`immediate-succs`, $\mathcal{C}_g$ and $\mathcal{C}_{g_2}$ are incomparable) **then**
      $\mathcal{R}$.`immediate-succs` = $\mathcal{R}$.`immediate-succs` $\cup \{g\}$;
    end if
  end for
end for

3.3 Generic association rule basis extraction

In this step, PRINCE extracts the *valid informative association rules*. For this purpose and using Proposition 2, PRINCE finds the frequent closed itemset corresponding to each equivalence class.

Proposition 2. *[15] Let f and f_1 be two closed itemsets such that f covers f_1 in the Galois lattice $\mathcal{L}_{\mathcal{C}_\mathcal{K}}$. Let MG_f be the set of minimal generators of f. The closed itemset f can be composed as follows: $f = \cup\{g | g \in MG_f\} \cup f_1$.*

The traversal of the *minimal generator lattice* is carried out in an ascending manner from the equivalence class whose frequent minimal generator is the empty set[7] (denoted $\mathcal{C}_\emptyset$) to the non subsumed equivalence class(es). If the closure of the empty set is not null, the exact generic association rule between the empty set and its closure is then extracted. Having the partial ordered structure built, PRINCE extracts the valid approximate generic association rules between the empty set and the frequent closed itemsets of the upper cover of $\mathcal{C}_\emptyset$. These closures are found, by applying Proposition 2, using the minimal generators of each equivalence class and the closure of the empty set. Equivalence classes forming the upper cover of $\mathcal{C}_\emptyset$ are stored which makes it possible to apply the same process to them. By the same manner, PRINCE treats higher levels of the *minimal generator lattice* until reaching the maximal equivalence class(es).

The pseudo-code of this step is given by the procedure GEN-GRB (Algorithm 2). We use the same notations of the procedure GEN-ORDER to which we add the field `FCI` to each element of $\mathcal{FMG}_\mathcal{K}$. Thus, for each frequent minimal generator g, this field allows to store the frequent closed itemset corresponding to $\mathcal{C}_g$ if g is its representative. In the GEN-GRB procedure, L_1 indicates the list of equivalence classes from which are extracted the valid informative association rules. By L_2, we note the list of equivalence classes which cover those forming L_1[8].

Example 1. Let us consider the extraction context $\mathcal{K}$ given by Figure 1 (Left) for *minsup*=2 and *minconf*=0.5. The first step allows the determination of the empty set closure, the sorted set $\mathcal{FMG}_\mathcal{K}$ and the negative border of minimal generators $\mathcal{GB}\text{d}^-$. Thus, $\emptyset''$=E, $\mathcal{FMG}_\mathcal{K}$ = {(∅,5), (C,4), (D,4), (A,3), (B,3), (F,3), (G,3), (CD,3), (AG,2), (BC,2), (BD,2), (DG,2), (FG,2)} and $\mathcal{GB}\text{d}^-$={(AB,1), (BF,1), (BG,1), (BCD,1)}. During the second step, PRINCE processes the element of $\mathcal{FMG}_\mathcal{K}$ by comparing each frequent minimal generator g, of size k ($k \geq 1$), with the immediate successor lists of its (k-1)-subsets. Since the list of immediate successors of the empty set is empty, C is added to ∅.`immediate-succs`. Then, D is compared to C. Since CD is a minimal generator, $\mathcal{C}_C$ and $\mathcal{C}_D$ are then incomparable and D is added to ∅.`immediate-succs`. A is then compared to this

[7] This class is called the Bottom element of the lattice [16]. The corresponding closure is calculated in the first step by collecting items appearing in all transactions of the extraction context.

[8] A test is carried out to check that an equivalence class does not belong to L_2. This test consists in checking if the corresponding frequent closed itemset were already calculated (Line 11 in Algorithm 2).

Algorithm 2 GEN-GRB

Require: The *minimal generator lattice* and the minimum threshold of confidence *minconf*.

Ensure: The corresponding frequent closed itemset of each equivalence class, the generic basis for exact association rules GB and the transitive reduction of the informative basis RI.

1: GB=$\emptyset$;
2: RI=$\emptyset$;
3: L_1={$\emptyset$};
4: L_2=$\emptyset$;
5: **while** ($L_1 \neq \emptyset$) **do**
6: **for all** ($g \in L_1$) **do**
7: **if** (g.FCI $\neq g$) **then**
8: $GB = GB \cup \{(t \Rightarrow (g.\texttt{FCI} - t), g.\texttt{support}) \mid t \in \mathcal{FMG}_{\mathcal{K}}$ and $t \in \mathcal{C}_g\}$;
9: **end if**
10: **for all** $g_1 \in g$.immediate-succs **do**
11: **if** (g_1.FCI=$\emptyset$) **then**
12: g_1.FCI=$\cup$ $\{t \in \mathcal{FMG}_{\mathcal{K}} \mid t \in \mathcal{C}_{g_1}\} \cup g$.FCI;
13: L_2=$L_2 \cup \{g_1\}$;
14: **end if**
15: **if** ((g_1.support/g.support) $\geq$ *minconf*) **then**
16: $RI = RI \cup \{(t \Rightarrow (g_1.\texttt{FCI} - t), g_1.\texttt{support}, g_1.\texttt{support}/g.\texttt{support}) \mid t \in \mathcal{FMG}_{\mathcal{K}}$ and $t \in \mathcal{C}_g\}$;
17: **end if**
18: **end for**
19: **end for**
20: L_1= L_2;
21: L_2= $\emptyset$;
22: **end while**

list. By comparing A to C, A.support < C.support and A.support = Supp(AC) and $\mathcal{C}_A$ is then a successor of $\mathcal{C}_C$. A is added to C.immediate-succs without any comparison since this list is still empty. A is also added to D.immediate-succs since A.support < D.support and A.support = Supp(AD). At this moment of processing, we have $\emptyset$.immediate-succs = {C,D} and B is added to this list since $\mathcal{C}_B$ is incomparable with $\mathcal{C}_C$ (BC is a minimal generator) and $\mathcal{C}_D$ (BD is also a minimal generator). F is then introduced into the *minimal generator lattice* by comparing it with the immediate successor list of its unique 0-subset, *i.e.*, the empty set. By comparing F to C, F.support < C.support and F.support = Supp(CF) and then $\mathcal{C}_F$ is a successor of $\mathcal{C}_C$. F is then compared to C.immediate-succs which contains A. F.support = A.support = Supp(AF) and thus F $\in \mathcal{C}_A$ whose A is the representative one. The MANAGE-EQUIV-CLASS function is then applied by replacing occurrences of F, in the immediate successor lists, by A (in this case, there is no occurrence) and by continuing comparisons with A instead of F (in this case, there are no more comparisons to do with F). G is then compared to $\emptyset$.immediate-succs equal to {C,B,D}. $\mathcal{C}_G$ is a successor of $\mathcal{C}_C$ since G.support < C.support and G.support = Supp(CG). After

comparing G with C.`immediate-succs` which only contains A, G is added to C.`immediate-succs` since $\mathcal{C}_G$ is incomparable with $\mathcal{C}_A$ (AG is a minimal generator). By comparing G to D (resp. B), $\mathcal{C}_G$ is incomparable with $\mathcal{C}_D$ (resp. $\mathcal{C}_B$) since DG (resp. BG) is a minimal generator. Then, CD is compared to the immediate successor lists of its 1-subsets, *i.e.*, C and D. $\mathcal{C}_C$ has $\mathcal{C}_A$ and $\mathcal{C}_G$ as immediate successors. By comparing CD and A, CD is affected to $\mathcal{C}_A$ since CD.`support` = A.`support` = Supp(ACD). The MANAGE-EQUIV-CLASS function is then applied. In particular, comparisons to carry out with CD will be made with A. A is then compared to the immediate successor list of the second 1-subset of CD, *i.e.*, D. However, D.`immediate-succs` contains only A and the comparison process stops. It is the same for the remainder of $\mathcal{FMG}_{\mathcal{K}}$. Having the *minimal generator lattice* built (*cf.* Figure 1 (Center)), an ascending sweeping is carried out from $\mathcal{C}_\emptyset$. As $\emptyset''$=E, the exact generic association rule $\emptyset \Rightarrow$ E is then extracted. $\emptyset$.`immediate-succs`={C,D,B}. The frequent closed itemset associated to $\mathcal{C}_C$ is then found and is equal to CE. The approximate generic association rule $\emptyset \Rightarrow$ CE, of a support equal to 4 and a confidence equal to 0.8, will be extracted. It is the same for $\mathcal{C}_D$ and $\mathcal{C}_B$. Using the same process and from $\mathcal{C}_C$, $\mathcal{C}_D$ and $\mathcal{C}_B$, the traversal of the *minimal generator lattice* is performed in an ascending way until extracting all valid informative association rules. The resulting generic association rule bases are sketched by Figure 2.

Exact generic association rules	
R_1: $\emptyset \Rightarrow$ E	R_8: G $\Rightarrow$ CE
R_2: C $\Rightarrow$ E	R_9: BC $\Rightarrow$ E
R_3: D $\Rightarrow$ E	R_{10}: BD $\Rightarrow$ E
R_4: B $\Rightarrow$ E	R_{11}: AG $\Rightarrow$ CDEF
R_5: A $\Rightarrow$ CDEF	R_{12}: DG $\Rightarrow$ ACEF
R_6: F $\Rightarrow$ ACDE	R_{13}: FG $\Rightarrow$ ACDE
R_7: CD $\Rightarrow$ AEF	

Approximate generic association rules	
R_{14}: $\emptyset \overset{0.8}{\Rightarrow}$ CE	R_{21}: D $\overset{0.5}{\Rightarrow}$ BE
R_{15}: $\emptyset \overset{0.8}{\Rightarrow}$ DE	R_{22}: B $\overset{0.66}{\Rightarrow}$ CE
R_{16}: $\emptyset \overset{0.6}{\Rightarrow}$ BE	R_{23}: B $\overset{0.66}{\Rightarrow}$ DE
R_{17}: C $\overset{0.75}{\Rightarrow}$ ADEF	R_{24}: A $\overset{0.66}{\Rightarrow}$ CDEFG
R_{18}: C $\overset{0.75}{\Rightarrow}$ GE	R_{25}: F $\overset{0.66}{\Rightarrow}$ ACDEG
R_{19}: C $\overset{0.5}{\Rightarrow}$ BE	R_{26}: CD $\overset{0.66}{\Rightarrow}$ AEFG
R_{20}: D $\overset{0.75}{\Rightarrow}$ ACEF	R_{27}: G $\overset{0.66}{\Rightarrow}$ ACDEF

Fig. 2. Left: *GB* basis. **Right:** *RI* basis.

3.4 Correctness and Computational cost

In this section, we prove the correctness of PRINCE algorithm and we evaluate its computational cost in the worst case.

Theorem 1. ***(correctness)*** *The* PRINCE *algorithm extracts all frequent minimal generators and derives all frequent closed itemsets and all valid informative association rules.*

Proof. During the first step, a minimal generator candidate c is pruned only if its estimate support is equal to its actual support or if it does not verify

the ideal order of minimal generators. Otherwise, c is a minimal generator and by comparing its actual support to *minsup*, PRINCE algorithm adds it to the frequent minimal generator set $\mathcal{FMG}_{\mathcal{K}}$ or to the negative border of minimal generators $\mathcal{GB}\mathrm{d}^{-}$. Thus, at the end of the first step of PRINCE, all frequent minimal generators are extracted in addition to the negative border of minimal generators.

During the second step, PRINCE takes care to introduce all frequent minimal generators into the *minimal generator lattice*. Indeed, a frequent minimal generator g is compared to the immediate successor list of all its (k-1)-subsets. The REPRESENTATIVE function allows to find the representative itemset of the equivalence class of a (k-1)-subset of g. Once the representative found, the used Proposition 1 treats both possible cases. The MANAGE-EQUIV-CLASS function is used only if g is compared to the representative of $\mathcal{C}_g$. At the end of this step, the *minimal generator lattice* is completely built.

During the third step, all equivalence classes are taken in consideration when deriving frequent closed itemsets and valid informative association rules. Indeed, each equivalence class $\mathcal{C}$, except $\mathcal{C}_{\emptyset}$, has *at least one immediate predecessor*. Hence, the representative of $\mathcal{C}$ belongs at least to one immediate successor list of another equivalence class, say $\mathcal{C}_1$. When treating $\mathcal{C}_1$, the frequent closed itemset of $\mathcal{C}$ is derived and $\mathcal{C}$ is added to the equivalence class list from which valid informative association rules will be derived in the next iteration. Thus, at the end of this step, all frequent closed itemsets and all valid informative association rules are derived.

Proposition 3. ***(computational cost)*** *In the worst case, the time complexity of* PRINCE *is* $O((n^3 + m) \times 2^n)$*, where* n *(resp.* m*) is the number of distinct items (resp. transactions) in the extraction context.*

Proof. The worst case is obtained when each extracted frequent itemset is a frequent closed minimal generator. Thus, the frequent itemset lattice strictly overlaps both the *Iceberg Galois lattice* and the *minimal generator* lattice. The number of frequent closed minimal generators is then equal to 2^n. We consider that each transaction contains the n distinct items.

During the first step, PRINCE performs two main tasks. The first task consists in candidate support computations and is of order $O(m \times 2^n)$. The second task consists in trying to prune non-minimal generator candidates and it is done in the order of $O(n^2 \times 2^n)$. The cost of the first step is then of order $O((n^2+m) \times 2^n)$.

During the second step, and for each frequent minimal generator g of size k, PRINCE performs, in the worst case, $O(k \times (n-k))$ comparisons ($(k \times (n-k))$ will be over-estimated by n^2). Indeed, the number of its (k-1)-subset is equal to k. Each (k-1)-subset g_1 has, in the worst case, $(n-k)$ immediate successors when comparing g with g_1.`immediate-succs`. Each comparison is performed by making the union of g with an element of g_1.`immediate-succs`. The union cost is $O(n)$. The search of the support of the itemset, result of this union, costs $O(n)$ since it is a minimal generator. The cost of the second step is then $O((n+n) \times n^2 \times 2^n)$, *i.e.*, $O(n^3 \times 2^n)$.

During the third step, and for each equivalence class $\mathcal{C}$, PRINCE performs two main tasks. The first task consists in deriving the corresponding frequent closed itemset f. This is carried out by performing the union of the set of frequent minimal generators of f, containing only one element, and a frequent closed itemset f_1, which is an immediate predecessor of f. The first task then costs $O(n)$. The second task consists in deriving valid informative association rules. As each frequent minimal generator is also closed, there is no exact generic association rules. However, by fixing *minconf* to 0, there are k approximate generic association rules, for an equivalence class whose frequent closed minimal generator is of size k. To derive each approximate generic association rule, PRINCE performs the difference between the frequent closed itemset f and the corresponding premise and this costs $O(n)$. The second task then costs $O(k \times n)$ (k will be over-estimated by n). Hence, the cost of the third step is $O((n + n^2) \times 2^n)$, *i.e.*, $O(n^2 \times 2^n)$.

Thus, in the worst case, the time complexity of PRINCE is the sum of costs of its three steps and is of order $O((n^3 + m) \times 2^n)$.

It is important to mention that although PRINCE constructs the partial order, its running time remains of the same order of magnitude as that of algorithms dedicated to the extraction of frequent closed itemsets [17].

4 Experimental results

In this section, we shed light on PRINCE performances vs those of CLOSE, A-CLOSE and TITANIC algorithms. PRINCE was implemented in the C language using gcc version 3.3.1. All experiments were carried out on a PC with a 2.4 GHz Pentium IV and 512 MB of main memory (with 2 GB of Swap) and running S.U.S.E Linux 9.0.

In all our experiments, all times reported are real times, including system and user times, into benchmark datasets (dense and sparse[9]) and "worst case" datasets. Figure 3 (Left) summarizes the characteristics of benchmark datasets. The definition of a "worst case" context is given as follows:

Definition 3. *A "worst case" context is a context $\mathcal{K} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$ where $\mathcal{O}$ represents a finite set of objects (or transactions) of size $(n+1)$, $\mathcal{A}$ is a finite set of attributes (or items) of size n and $\mathcal{R}$ is a binary (incidence) relation (i.e., $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$). Each object, among the first n ones, is verified by $(n\text{-}1)$ distinct attributes. The last object is verified by all attributes. Each attribute is checked by n distinct objects.*

Thus, in a "worst case" dataset, each closed itemset is equal to its (minimal) generator. Hence, from a "worst case" dataset of dimension equal to $(n+1)\times n$, 2^n frequent closed itemsets can be extracted when *minsup* is fixed to 1 transaction. Figure 3 (Right) presents an example of a "worst case" dataset for n=4.

[9] All these datasets are downloadable on the following address: *http://fimi.cs.helsinki.fi/data.*

Dataset	Type	# items	Avg. tr. size	# transactions
Mushroom	dense	119	23	8124
T40I10D100K	sparse	1000	40	100000

	i_1	i_2	i_3	i_4
1		×	×	×
2	×		×	×
3	×	×		×
4	×	×	×	
5	×	×	×	×

Fig. 3. (**Left**) Benchmark dataset characteristics. (**Right**) A "worst case" dataset for n=4.

Figure 4 shows execution times of PRINCE[10] algorithm compared to those of CLOSE, A-CLOSE and TITANIC algorithms.

- **Mushroom**: In the case of Mushroom dataset, PRINCE performances are better than those of CLOSE, A-CLOSE and TITANIC for all *minsup* values, given the important role played by equivalence class management functions. Indeed, for a value of *minsup* equal to 0.1%, the number of frequent minimal generators (equal to 360,166) is almost to 2.2 times the number of frequent closed itemsets (equal to 164,117). TITANIC performances decrease in a significant way due to the extension attempts carried out for each frequent minimal generator. Indeed, for *minsup* = 0.1%, 116 items, among 119, are frequent and the maximum size of a frequent minimal generator is only equal to 10 items.

- **T40I10D100K**: PRINCE performances for this dataset are largely better than those of CLOSE, A-CLOSE and TITANIC for all *minsup* values. Thus, CLOSE and A-CLOSE are handicapped by a large average transaction size (40 items). In the same way, TITANIC performances regress considerably for the same reasons previously evoked. The comparison cost for a frequent minimal generator, in the case of PRINCE, being definitely more reduced than the intersection operations performed in CLOSE and A-CLOSE and the extension attempts elaborated in TITANIC, explains the big gap between PRINCE performances and those of remaining algorithms.

- **"Worst case" datasets**: For these experiments, *minsup* was fixed to 1 transaction. We tested 26 datasets showing the variation of n from 1 to 26. The execution times of the four algorithms began to be distinguishable only starting from the value of n equal to 15. The PRINCE algorithm performances remain better than those of CLOSE, A-CLOSE and TITANIC algorithms. CLOSE and TITANIC executions stop for n=24 for lack of memory space. It is the same for A-CLOSE for n=25 and PRINCE for n=26. It is important to mention that the partial order construction requires to store much more information than needed when aiming only to extract frequent closed itemsets. Thus, the use of only *one trie* to store information about all minimal generators instead of *several tries*, as in CLOSE, A-CLOSE and TITANIC algorithms[11], is an attempt aiming to reduce the memory need of PRINCE algorithm.

[10] The *minconf* value is set to 0.

[11] Indeed, in the case of these three algorithms, a *trie* is used to save information about each set of (frequent) minimal generators of size k.

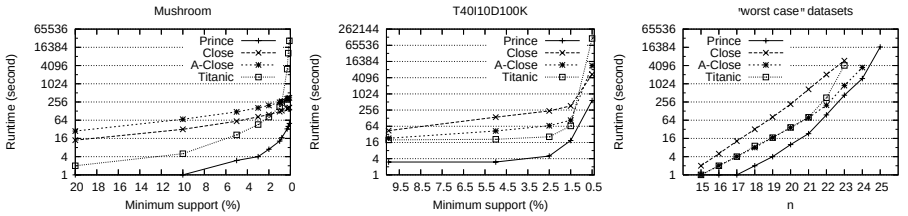

Fig. 4. PRINCE performances vs those of CLOSE, A-CLOSE and TITANIC.

5 Conclusion

In this paper, we proposed a new algorithm, called PRINCE, for an efficient extraction of frequent closed itemsets and their respective minimal generators as well as the generic association rule bases. To this end, PRINCE builds the partial order contrary to the existing algorithms. A main characteristic of PRINCE algorithm is that it relies only on minimal generators to build the underlying partial order. Carried out experiments outlined that PRINCE largely outperforms existing "Test-and-generate" algorithms of the literature for both benchmark and "worst case" contexts. In the near future, we plan to tackle two issues. Firstly, we plan to study the possibility of integrating the work of Calders *et al.* [18] in the first step of PRINCE. Indeed, this work can be applied to any set verifying the property of ideal order such as the set of frequent minimal generators in our case. Secondly, we propose to add constraints [19], so that the number of generic association rules will be reduced while keeping the most interesting for the user.

Acknowledgements The authors are deeply grateful to Yves Bastide who kindly accepted to provide source codes of CLOSE, A-CLOSE and TITANIC algorithms.

References

1. Pei, J., Han, J., Mao, R., Nishio, S., Tang, S., Yang, D.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In: Proceedings of the ACM-SIGMOD DMKD'00, Dallas, Texas, USA. (2000) 21–30
2. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient Mining of Association Rules Using Closed Itemset Lattices. Journal of Information Systems **24** (1999) 25–46
3. Wille, R.: Restructuring lattices theory: An approach based on hierarchies of concepts. I. Rival, editor, Ordered Sets, Dordrecht-Boston (1982) 445–470
4. BenYahia, S., Nguifo, E.M.: Approches d'extraction de règles d'association basées sur la correspondance de Galois. In Boulicault, J.F., Cremilleux, B., eds.: Revue d'Ingénierie des Systèmes d'Information (ISI), Hermès-Lavoisier. Volume 9. (2004) 23–55
5. Pasquier, N., Bastide, Y., Touil, R., Lakhal, L.: Discovering frequent closed itemsets. In Beeri, C., Buneman, P., eds.: Proceedings of 7th International Conference

on Database Theory (ICDT'99), LNCS, volume 1540, Springer-Verlag, Jerusalem, Israel. (1999) 398–416
6. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with TITANIC. Journal on Knowledge and Data Engineering (KDE) **2** (2002) 189–222
7. Zaki, M.J., Hsiao, C.J.: CHARM: An efficient algorithm for closed itemset mining. In: Proceedings of the 2nd SIAM International Conference on Data Mining, Arlington, Virginia, USA. (2002) 34–43
8. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In Goethals, B., Zaki, M.J., Bayardo, R., eds.: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04). Volume 126 of CEUR Workshop Proceedings, Brighton, UK. (2004)
9. Lucchesse, C., Orlando, S., Perego, R.: DCI-CLOSED: a fast and memory efficient algorithm to mine frequent closed itemsets. In Goethals, B., Zaki, M.J., Bayardo, R., eds.: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04). Volume 126 of CEUR Workshop Proceedings, Brighton, UK. (2004)
10. BenYahia, S., Cherif, C.L., Mineau, G., Jaoua, A.: Découverte des règles associatives non redondantes : application aux corpus textuels. Revue d'Intelligence Artificielle (special issue of Intl. Conference of Journées francophones d'Extraction et Gestion des Connaissances (EGC'2003)), Lyon, France **17** (2003) 131–143
11. Bastide, Y., Pasquier, N., Taouil, R., Lakhal, L., Stumme, G.: Mining minimal non-redundant association rules using frequent closed itemsets. In: Proceedings of the International Conference DOOD'2000, LNAI, volume 1861, Springer-Verlag, London, UK. (2000) 972–986
12. Kryszkiewicz, M.: Concise representations of association rules. In Hand, D.J., Adams, N., Bolton, R., eds.: Proceedings of Exploratory Workshop on Pattern Detection and Discovery in Data Mining (ESF), 2002, LNAI, volume 2447, Springer-Verlag, London, UK. (2002) 92–109
13. Kryszkiewicz, M.: Concise representation of frequent patterns based on disjunction-free generators. In: Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM), San Jose, California, USA. (2001) 305–312
14. Davey, B., Priestley, H.: Introduction to Lattices and Order. Cambridge University Press (2002)
15. Hamrouni, T., BenYahia, S., Slimani, Y.: PRINCE : Extraction optimisée des bases génériques de règles sans calcul de fermetures. In: Proceedings of the 23rd International Conference INFORSID, Inforsid Editions, Grenoble, France. (2005) 353–368
16. Ganter, B., Wille, R.: Formal Concept Analysis. Springer-Verlag (1999)
17. Pasquier, N.: Datamining: Algorithmes d'extraction et de réduction des règles d'association dans les bases de données. Thèse de doctorat, Ecole Doctorale Sciences pour l'Ingénieur de Clermont Ferrand, Université Clermont Ferrand II, France (2000)
18. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In Elomaa, T., Mannila, H., Toivonen, H., eds.: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery, PKDD 2002, LNCS, volume 2431, Springer-Verlag, Helsinki, Finland. (2002) 74–85
19. Bonchi, F., Lucchese, C.: On closed constrained frequent pattern mining. In: Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04), Brighton, UK. (2004) 35–42