Levelwise search of frequent patterns with counting inference

Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, Lotfi Lakhal

Levelwise Search of Frequent Patterns with Counting Inference

Yves Bastide[1], Rafik Taouil[2], Nicolas Pasquier[3],
Gerd Stumme[4], Lotfi Lakhal[5]

[1] LIMOS - CNRS FRE 2239, Université Blaise Pascal,
24 avenue des Landais, 63 177 Aubière cedex, France;
bastide@libd2.univ-bpclermont.fr

[2] INRIA LORRAINE,
615, rue du Jardin Botanique, 54 602 Villers Lès Nancy cedex, France;
taouil@loria.fr

[3] I3S - CNRS UPRESA 6070 - UNSA, 2000 route des Lucioles,
Algorithmes-Euclide B, BP 121, 06 903 Sophia-Antipolis, France;
Nicolas.Pasquier@unice.fr

[4] Institut für angewandte Informatik und Formale Beschreibungsverfahren,
Universität Karlsruhe (TH), D-76128 Karlsruhe, Germany;
stumme@aifb.uni-karlsruhe.de

[5] LIM - CNRS FRE 2246, Université de la Méditerranée,
case 901, 163 avenue de Luminy, 13 288 Marseille cedex 9, France;
lakhal@lim.univ-mrs.fr

Abstract

In this paper, we address the problem of the efficiency of the main phase of most data mining applications: The frequent pattern extraction. This problem is mainly related to the number of operations required for counting pattern supports in the database, and we propose a new method, called pattern counting inference, that allows to perform as few support counts as possible. Using this method, the support of a pattern is determined without accessing the database whenever possible, using the supports of some of its sub-patterns called key patterns. This method was implemented in the PASCAL algorithm that is an optimization of the simple and efficient Apriori algorithm. Experiments comparing PASCAL to the Apriori, Close and Max-Miner algorithms, each one representative of a frequent patterns discovery strategy, show that PASCAL improves the efficiency of the frequent pattern extraction from correlated data and that it does not induce additional execution times when data is weakly correlated.

Keywords: Data mining, frequent patterns extraction, pattern counting inference, key patterns, algorithms.

1 Introduction

Knowledge discovery in databases (KDD) is defined as the non-trivial extraction of valid, implicit, potentially useful and ultimately understandable information in large databases. For

several years, a wide range of applications in various domains have benefited from KDD techniques and many works have been conducted on this topic. The problem of mining frequent patterns first arose as a sub-problem of mining association rules [AS94]. A frequent pattern is a set of binary attributes (items) which support, i.e., the number of objects in the database "containing" it, is at least equal to a minimum support threshold *minsup* defined by the user. It then turned out that frequent patterns are involved in a variety of problems [HPY00]: mining sequential patterns [AS95], episodes [MTV97], correlations [BMS97, SBM98], multi-dimensional patterns [KHC97, LSW97], maximal patterns [Bay98, ZPOL97, LK98], closed patterns [PBTL99a, PBTL99b, Pei00, TPBL00]. The complexity of this problem is exponential in the size of the binary relation in input (database) and this relation has to be scanned several times during the process in order to count patterns supports. Thus, optimized algorithms for mining frequent patterns are required for real-life applications that are submitted to strong efficiency constraints.

1.1 Related Work

Three approaches have been proposed for mining frequent patterns: The first is traversing iteratively the set of all patterns in a levelwise manner [MT97]. During each iteration corresponding to a level, a set of candidate patterns is created by joining the frequent patterns discovered during the previous iteration, the supports of all candidate patterns are counted and infrequent ones are discarded. The most prominent algorithm based on this approach is the Apriori algorithm [AS94], that uses identical properties as the OCD algorithm [MTV94] proposed concurrently. A variety of modifications of this algorithm arose [BMUT97, GPW98, PCY95, SON95, Toi96] in order to improve different efficiency aspects. However, all of these algorithms have to determine the supports of *all* frequent patterns and of some infrequent ones from the database.

The second approach is based on the extraction of maximal[1] frequent patterns, from which all supersets are infrequent and all subsets are frequent. This approach combines a levelwise bottom-up traversal with a top-down traversal in order to quickly find the maximal frequent patterns. Then, all frequent patterns are derived from these ones and one last database scan is carried on to count their support. The most prominent algorithm using this approach is Max-Miner [Bay98]. Experimental results have shown that this approach is particularly efficient for extracting maximal frequent patterns, but when applied for extracting all frequent patterns performances drastically decrease because of the cost of the last scan which requires roughly an inclusion test between each frequent pattern and each object of the database. As for algorithms based on the first approach, algorithms based on this approach have to extract the supports of all frequent patterns from the database.

The third approach, represented by the Close algorithm [PBTL99a], is based on the theoretical framework introduced in [PBTL98] that uses the closure of the Galois connection [GW99]. In this approach, the frequent closed patterns (and their support) are extracted from the database in a levelwise manner. A closed pattern is the greatest pattern common to a set of objects of the database; each non-closed pattern has the same properties (same set of objects containing it, and thus an identical support) as its closure, i.e. the smallest closed pattern containing it. Then, all frequent patterns as well as their support are derived from the frequent closed patterns and their support without accessing the database. Hence not all patterns are considered during the most expensive part of the algorithm (counting the supports of the patterns) and the search

[1] 'Maximal' means 'maximal with respect to set inclusion'.

space is drastically reduced, especially for correlated data. Experiments have shown that this approach is much more efficient than the two previous ones on such data.

1.2 Contribution

We propose a new method, called *pattern counting inference*, that minimizes as much as possible the number of pattern support counts performed when extracting frequent patterns. This method relies on the concept of *key patterns*, where a key pattern is a minimal (with respect to set inclusion) pattern of an *equivalence class* gathering all patterns common to the same objects of the database relation[2]. Hence, all patterns in an equivalence class have the same support and the supports of the non-key patterns of an equivalence class can be determined using the supports of the key patterns of this class. With pattern counting inference, only the supports of the frequent key patterns (and some infrequent ones) are determined from the database, while supports of the frequent non-key patterns are derived from those of the frequent key patterns.

This method was implemented in the PASCAL [3] algorithm that is an optimization of the simple and efficient Apriori algorithm. In PASCAL, as in Apriori, frequent patterns are extracted in a levelwise manner: During each iteration, candidate patterns of size k are created by joining the frequent patterns of size $k-1$, their supports are determined and infrequent ones are discarded. Using counting inference, if a candidate pattern of size k is a non-key pattern, then its support is equal to the minimal support among the patterns of size $k-1$ that are its subsets. This allows to reduce the number of patterns considered during each database pass while counting supports and, even more important, to reduce the total number of passes. This optimization is valid since key patterns have a property that is compatible with the pruning of Apriori: all subsets of a key pattern are key patterns and all supersets of a non-key pattern are non-key patterns. Moreover, in comparison to most other modifications of Apriori, this induces a minimal impact on the understandability and simplicity of the algorithm implementation. The important difference is to determine as much support counts as possible without accessing the database by information gathered in previous passes.

We conducted experiments to compare PASCAL to the Apriori, Close and Max-Miner algorithms, each one representative of a frequent patterns discovery strategy, onto several datasets. Results show that PASCAL clearly improves the efficiency of the frequent pattern extraction from correlated data and that counting inference does not induce any execution overtime when data is weakly correlated.

1.3 Organization of the Paper

In the next section, we recall the problem of mining frequent patterns. The essential notions and the definitions of key patterns and pattern counting inference are given in Section 3. The PASCAL algorithm is described in Section 4 and experimental results for comparing its efficiency to those of Apriori, Max-Miner and Close are presented in Section 5. A summary of the paper and some perspectives are given in Section 6.

[2] A similar notion of equivalence classes was also recently proposed by R. Bayardo and R. Agrawal [Bay99] to characterize "a-maximal" rules (i.e., association rules with maximal antecedent).

[3] The French mathematician Blaise Pascal (* Clermont-Ferrand 1623, † 1662 Paris) invented an early computing device.

2 The Problem of Mining Frequent Patterns

Definition 1. Let $\mathbb{P}$ be a finite set of *items*, $\mathbb{O}$ a finite set of *objects* (e. g., transaction ids) and $\mathbb{R} \subseteq \mathbb{O} \times \mathbb{P}$ a binary relation (where $(o, p) \in \mathbb{R}$ may be read as "item p is included in transaction o"). The triple $\mathbb{D} = (\mathbb{O}, \mathbb{P}, \mathbb{R})$ is called *dataset*.

Each subset P of $\mathbb{P}$ is called a *pattern*. We say that a pattern P is *included* in an object $o \in \mathbb{O}$ if $(o, p) \in \mathbb{R}$ for all $p \in P$. Let f be the function which assigns to each pattern $P \subseteq \mathbb{P}$ the set of all objects which include this pattern: $f(P) = \{o \in \mathbb{O} \mid o \text{ includes } P\}$.

The *support* of a pattern P is given by $\text{supp}(P) = \text{card}(f(P))/\text{card}(\mathbb{O})$. For a given threshold $\text{minsup} \in [0, 1]$, a pattern P is called *frequent pattern* if $\text{supp}(P) \geq \text{minsup}$.

Problem: The task of mining frequent patterns consists in determining all frequent patterns together with their supports[4] for a given threshold minsup.

3 Key Patterns and Pattern Counting Inference

In this section, we give the theoretical basis of the new PASCAL algorithm. This basis provides at the same time the proof of correctness of the algorithm. In Section 4, these theorems will be turned into pseudo-code.

Like Apriori, PASCAL traverses the powerset of $\mathbb{P}$ levelwise: At the k^{th} iteration, the algorithm generates first all *candidate k-patterns*.

Definition 2. A k-pattern P is a subset of $\mathbb{P}$ with $\text{card}(P) = k$. A *candidate k-pattern* is a k-pattern where all its proper sub-patterns are frequent.

For the candidate k-patterns one database pass is used to determine their support. Then infrequent patterns are pruned. This approach works because of the well-known fact that a pattern cannot be frequent if it has an infrequent sub-pattern.

3.1 Key Patterns

Our approach is based on the observation that patterns can be considered as "equivalent" if they are included in exactly the same objects. We describe this fact by the following equivalence relation θ on patterns.

Definition 3. For patterns $P, Q \subseteq \mathbb{P}$, we let $P \,\theta\, Q$ if and only if $f(P) = f(Q)$. The set of patterns which are *equivalent* to a pattern P is given by $[P] = \{\, Q \subseteq \mathbb{P} \mid P \,\theta\, Q \,\}$.

In the case of patterns P and Q with $P \,\theta\, Q$, both patterns obviously have the same support:

Lemma 1. *Let P and Q be two patterns.*
(i) $P \,\theta\, Q \implies \text{supp}(P) = \text{supp}(Q)$
(ii) $P \subseteq Q \wedge \text{supp}(P) = \text{supp}(Q) \implies P \,\theta\, Q$

Proof. *(i)* $P \,\theta\, Q \iff f(P) = f(Q) \implies \text{supp}(P) = \text{card}(f(P))/\text{card}(\mathbb{O}) = \text{card}(f(Q))/\text{card}(\mathbb{O}) = \text{supp}(Q)$.

[4] There exist also applications where the supports need not be known exactly. We only consider the case where all supports have to be determined as well.

(*ii*) Since $P \subseteq Q$ and f is monotonous decreasing[5], we have $f(P) \supseteq f(Q)$. $\text{supp}(P) = \text{supp}(Q)$ is equivalent to $\text{card}(f(P)) = \text{card}(f(Q))$ which implies with the former $f(P) = f(Q)$ and thus $P \,\theta\, Q$. □

Hence if we knew the relation θ in advance, we would need to count the support of only one pattern in each equivalence class. Of course we do not know the relation in advance, but we can construct it step by step[6]. Thus, we will (in general) need to determine the support of more than one pattern in each class, but not of all of them. If we already have determined the support of a pattern P in the database and pass later a pattern $Q \in [P]$, then we need not access the database for it because we know that $\text{supp}(Q) = \text{supp}(P)$.

The first patterns of an equivalence class that we reach using a levelwise approach are exactly the minimal[7] patterns in the class:

Definition 4. A pattern P is a *key pattern* if $P \in \min[P]$. A *candidate key pattern* is a pattern where all its proper sub-patterns are frequent key patterns.

Observe that all candidate key patterns are also candidate patterns.

3.2 Pattern Counting Inference

In the algorithm, we apply the pruning strategy to both candidate patterns and candidate key patterns. This is justified by the following theorem.

Theorem 2. (*i*) *If Q is a key pattern and $P \subseteq Q$, then P is also a key pattern.*
(*ii*) *If P is not a key pattern and $P \subseteq Q$, then Q is not a key pattern either.*[8]

Proof. (*ii*) Let $P \subseteq Q$ and P is not a key pattern. Then, $\exists P' \in \min[P]$ with $P' \subset P$. From $f(P') = f(P)$ it follows that $f(Q) = f(Q \setminus (P \setminus P'))$. Hence Q is not minimal in $[Q]$ and thus by definition not a key pattern. (*i*) is a direct logical consequence of (*ii*). □

The algorithm determines, at each iteration, the key patterns among the candidate key patterns by using (*ii*) of the following theorem:

Theorem 3. *Let P be a pattern.*
(*i*) *Let $p \in P$. Then $P \in [P \setminus \{p\}]$ if and only if* $\text{supp}(P) = \text{supp}(P \setminus \{p\})$.
(*ii*) *P is a key pattern if and only if* $\text{supp}(P) \neq \min_{p \in P}(\text{supp}(P \setminus \{p\}))$.

Proof. (*i*) The "if" part follows from Lemma 1 (*ii*). The "only if" part is obvious. (*ii*) From (*i*) we deduce that P is a key pattern if and only if $\text{supp}(P) \neq \text{supp}(P \setminus \{p\})$, for all $p \in P$. Since supp is a monotonous decreasing function, this is equivalent to (*ii*). □

Since all candidate key patterns are also candidate patterns, when generating all candidate patterns for the next level we can at the same time determine the candidate key patterns among them.

---

[5] Given a set of items included in some objects, its supersets will be included in at most the same objects.

[6] In the algorithm, the equivalence relation is not explicitly generated, but is – as the algorithm is based on the following theorems – implicitly used.

[7] 'Minimal' means 'minimal with respect to set inclusion'.

[8] In mathematical terms, (*i*) and (*ii*) state that the set of key patterns is an order ideal (or down-set) of $(2^{\mathbb{P}}, \subseteq)$.

If we reach a candidate k-pattern which is not a candidate key pattern, then we already passed along at least one of the key patterns in its equivalence class in an earlier iteration. Hence we already know its support. Using the following theorem, we determine this support *without accessing the database*:

Theorem 4. *If P is a non-key pattern, then*

$$\mathrm{supp}(P) = \min_{p \in P}(\mathrm{supp}(P \setminus \{p\})).$$

Proof. "$\leq$" follows from the fact that supp is a monotonous decreasing function. "$\geq$": If P is not a key pattern then there exists $p \in P$ with $P \,\theta\, P \setminus \{p\}$. Hence $\mathrm{supp}(P) = \mathrm{supp}(P \setminus \{p\}) \geq \min_{q \in P}(\mathrm{supp}(P \setminus \{q\}))$. □

Thus the database pass needs to count the supports of the candidate key patterns only. Knowing this, we can summarize PASCAL as follows: It works exactly as Apriori, but counts only those supports in the database pass which cannot be derived from supports already computed. Thus we can, on each level, restrict the expensive count in the database to some of the candidates. But even better, from some level on, all candidate pattern may be known to be non-key patterns. Then all remaining frequent patterns and their support can be derived without accessing the database any more. In the worst case (i. e., in weakly correlated data), all candidate patterns are also candidate key patterns. Then the algorithm behaves exactly as Apriori with no significant overhead.

4 The PASCAL algorithm

In this section, we transform the theorems given in the last section into an algorithm. The pseudo-code is given in Algorithm 1. A list of notations is provided in Table 1. We assume that $\mathbb{P}$ is linearly ordered, e. g., $\mathbb{P} = \{1, \ldots, n\}$. This will be used in PASCAL-GEN.

k	is the counter which indicates the current iteration. In the kth iteration, all frequent k-patterns and all key patterns among them are determined.
$\mathcal{P}_k$	contains after the kth iteration all frequent k-patterns P together with their support P.supp, and a boolean variable P.key indicating if P is a (candidate) key pattern.
$\mathcal{C}_k$	stores the candidate k-patterns together with their support (if known), the boolean variable P.key, and a counter P.pred_supp which stores the minimum of the supports of all $(k-1)$-sub-patterns of P.

Table 1: Notations used in PASCAL

The algorithm starts with the empty set, which always has a support of 1 and which is (by definition) a key pattern (step 1 and 2). In step 3, frequent 1-patterns are determined. They are marked as key patterns unless their support is 1 (steps 4–6). The main loop is similar to the one in Apriori (steps 7 to 21). First, PASCAL-GEN is called to compute the candidate patterns. The support of key ones is determined via a database pass (steps 10–14).

Then (steps 15–20) the 'traditional' pruning (step 16) is done. At the same time, for all remaining candidate key patterns, it is determined whether they are key or not (step 17 and 18).

Algorithm 1 PASCAL

```
1)  ∅.supp ← 1; ∅.key ← true;
2)  P_0 ← {∅};
3)  P_1 ← {frequent 1-patterns};
4)  forall p ∈ P_1 do begin
5)    p.pred_supp ← 1; p.key ← (p.supp ≠ 1);
6)  end;
7)  for (k = 2; P_{k-1} ≠ ∅; k++) do begin
8)    C_k ← PASCAL-GEN(P_{k-1});
9)    if ∃c ∈ C_k | c.key then
10)     forall o ∈ D do begin
11)       C_o ← subset(C_k, o);
12)       forall c ∈ C_o | c.key do
13)         c.supp++;
14)     end;
15)   forall c ∈ C_k do
16)     if c.supp ≥ minsup then begin
17)       if c.key and c.supp = c.pred_supp then
18)         c.key ← false;
19)       P_k ← P_k ∪ {c};
20)     end;
21) end;
22) return ⋃_k P_k.
```

The way PASCAL-GEN operates is basically known from the generator function Apriori-Gen which was introduced in [AS94]. When called at the kth iteration, it uses as input the set of frequent $(k-1)$-patterns $\mathcal{P}_{k-1}$. Its output is the set of candidate k-patterns. In addition to Apriori-Gen's *join* and *prune* steps, PASCAL-GEN makes the new candidates inherit the fact of being or not a candidate key pattern (step 9) by using Theorem 2; and it determines at the same time the support of all non key candidate patterns (step 12) by using Theorem 4.

Running example. We illustrate the PASCAL algorithm on the following dataset for minsup = 2/5:

Object	Items				
1	A	C	D	F	
2	B	C	E	F	
3	A	B	C	E	F
4	B	E	F		
5	A	B	C	E	F

The algorithm performs first one database pass to count the support of the 1-patterns. The candidate pattern $\{D\}$ is pruned because it is infrequent. As $\{F\}$ has the same support as the empty set, $\{F\}$ is marked as a non-key pattern:

Algorithm 2 PASCAL-GEN

Input: $\mathcal{P}_{k-1}$, the set of frequent $(k-1)$-patterns p with their support p.supp and the p.key flag.
Output: $\mathcal{C}_k$, the set of candidate k-patterns c each with the flag c.key, the value c.pred_supp, and the support c.supp if c is not a key pattern.

```
1) insert into C_k
   select p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}
   from P_{k-1} p, P_{k-1} q
   where p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1};
2) forall c ∈ C_k do begin
3)    c.key ← true; c.pred_supp ← +∞;
4)    forall (k-1)-subsets s of c do begin
5)       if s ∉ P_{k-1} then
6)          delete c from C_k;
7)       else begin
8)          c.pred_supp ← min(c.pred_supp, s.supp);
9)          if not s.key then c.key ← false;
10)      end;
11)   end;
12)   if not c.key then c.supp ← c.pred_supp;
13) end;
14) return C_k.
```

$\mathcal{P}_1$	supp	key
$\{A\}$	3/5	t
$\{B\}$	4/5	t
$\{C\}$	4/5	t
$\{E\}$	4/5	t
$\{F\}$	1	f

At the next iteration, all candidate 2-patterns are created and stored in $\mathcal{C}_2$. At the same time, the support of all patterns containing $\{F\}$ as sub-pattern is computed. Then a database pass is performed to determine the supports of the remaining six candidate patterns:

$\mathcal{C}_2$	pred_supp	key	supp
$\{AB\}$	3/5	t	?
$\{AC\}$	3/5	t	?
$\{AE\}$	4/5	t	?
$\{AF\}$	3/5	f	3/5
$\{BC\}$	4/5	t	?
$\{BE\}$	4/5	t	?
$\{BF\}$	4/5	f	4/5
$\{CE\}$	4/5	t	?
$\{CF\}$	4/5	f	4/5
$\{EF\}$	4/5	f	4/5

$\mathcal{P}_2$	supp	key
$\{AB\}$	2/5	t
$\{AC\}$	3/5	f
$\{AE\}$	2/5	t
$\{AF\}$	3/5	f
$\{BC\}$	3/5	t
$\{BE\}$	4/5	f
$\{BF\}$	4/5	f
$\{CE\}$	3/5	t
$\{CF\}$	4/5	f
$\{EF\}$	4/5	f

At the third iteration, it turns out in PASCAL-GEN that each newly generated candidate pattern contains at least one sub-pattern which is not a key pattern. Hence all new candidate patterns are not candidate key patterns. All their supports are determined directly in PASCAL-GEN. *From there, the database will not be accessed any more.*

$\mathcal{C}_3$	pred_supp	key	supp
$\{ABF\}$	2/5	f	2/5
$\{ABC\}$	2/5	f	2/5
$\{ABE\}$	2/5	f	2/5
$\{ACE\}$	2/5	f	3/5
$\{ACF\}$	3/5	f	3/5
$\{AEF\}$	2/5	f	2/5
$\{BCE\}$	3/5	f	3/5
$\{BCF\}$	3/5	f	3/5
$\{BEF\}$	4/5	f	4/5
$\{CEF\}$	3/5	f	3/5

$\mathcal{P}_3$	supp	key
$\{ABF\}$	2/5	f
$\{ABC\}$	2/5	f
$\{ABE\}$	2/5	f
$\{ACE\}$	2/5	f
$\{ACF\}$	3/5	f
$\{AEF\}$	2/5	f
$\{BCE\}$	3/5	f
$\{BCF\}$	3/5	f
$\{BEF\}$	4/5	f
$\{CEF\}$	3/5	f

In the fourth and fifth iteration, all supports are determined directly in PASCAL-GEN. In the sixth iteration, PASCAL-GEN generates no new candidate patterns, thus no frequent 6-patterns are computed and the algorithm stops:

$\mathcal{C}_4$	pred_supp	key	supp
$\{ABCE\}$	2/5	f	2/5
$\{ABCF\}$	2/5	f	2/5
$\{ABEF\}$	2/5	f	2/5
$\{ACEF\}$	2/5	f	3/5
$\{BCEF\}$	3/5	f	3/5

$\mathcal{P}_4$	supp	key
$\{ABCE\}$	2/5	f
$\{ABCF\}$	2/5	f
$\{ABEF\}$	2/5	f
$\{ACEF\}$	2/5	f
$\{BCEF\}$	3/5	f

$\mathcal{C}_5$	pred_supp	key	supp
$\{ABCEF\}$	2/5	f	2/5

$\mathcal{P}_5$	supp	key
$\{ABCEF\}$	2/5	f

Hence PASCAL needs two database passes in which the algorithm counted the supports of $6 + 6 = 12$ patterns. Apriori would have needed five database passes for counting the supports of $6 + 10 + 10 + 5 + 1 = 32$ patterns for the same dataset. All other current algorithms (with the only exception of Close) may need less than five passes, but they all have to perform the 32 counts.

5 Experimental Evaluation

We evaluated PASCAL against three algorithms, each representative of one frequent patterns discovery strategy: Apriori, Close, and Max-Miner. Max-Miner implementation was kindly provided by Roberto Bayardo, and retrieving the frequent patterns' support from the maximal frequent ones was done using a brute-force method[9]. PASCAL, Apriori, Close and this final step to Max-Miner all shared the same data structures and general organization. Optimizations such as special handling of pass two or items reordering were disabled.

Characteristics of the datasets used are given in Table 2. These datasets are the T20I6D100K, T25I10D10K and T25I20D100K[10] synthetic datasets that mimic market basket data, the C20-

[9] In the following tables, we distinguished the time spent by Max-Miner itself and the support retrieval step.

[10] `http://www.almaden.ibm.com/cs/quest/syndata.html`

D10K and C73D10K census datasets from the PUMS sample file[11], and the MUSHROOMS[12] dataset describing mushrooms characteristics [UCI99]. In all experiments, we attempted to choose significant minimum support threshold values.

Name	Number of objects	Average size of objects	Number of items
T20I6D100K	100,000	20	1,000
T25I10D10K	10,000	25	1,000
T25I20D100K	100,000	25	10,000
C20D10K	10,000	20	386
C73D10K	10,000	73	2,178
MUSHROOMS	8,416	23	128

Table 2: Datasets

Related work have shown that the behavior of algorithms for extracting frequent patterns depends mainly on the dataset characteristics. Weakly correlated data, such as synthetic data, constitute easy cases for the extraction since few patterns are frequent. For such data, all algorithms give acceptable response times as we can observe in Section 5.1 in which experimental results obtained for the T20I6D100K, T25I10D10K and T25I20D100K datasets are presented. On the contrary, correlated data constitute far more difficult cases for the extraction due to the important proportion of patterns that are frequent among all patterns. Such data represent a huge part of real-life datasets, and differences between extraction times obtained widely vary depending on the algorithm used. Experimental results obtained for the C20D10K, C73D10K and MUSHROOMS datasets, that are made up of correlated data, are given in Section 5.2.

5.1 Weakly correlated data

The T20I6D100K, T25I10D10K and T25I20D100K synthetic datasets are constructed according to the properties of market basket data that are typical weakly correlated data. In these datasets, the number of frequent patterns is small compared to the total number of patterns and, in most cases, nearly all the frequent patterns are also key patterns.

Response times for the T20I6D100K dataset are presented numerically in Table 3 and graphically in Figure 1. In this dataset, all frequents patterns are key patterns and Apriori and PASCAL behave identically and response times obtained from them and Max-Miner are similar. The Close algorithm gives higher response times due to the number of intersection operations needed for computing the closures of the candidate patterns.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
1.00	1,534	13.14	13.51	25.91	2.60	5.03
0.75	4,710	20.41	20.67	35.29	4.44	11.06
0.50	26,950	44.00	44.38	67.82	6.87	35.37
0.25	155,673	117.97	117.79	182.95	15.64	109.14

Table 3: Response times for T20I6D100K.

Results for the T25I20D100K dataset are presented numerically in Table 4 and graphically in Figure 2. For this dataset, nearly all frequent patterns are key patterns, and results are similar

[11] `ftp://ftp2.cc.ukans.edu/pub/ippbr/census/pums/pums90ks.zip`

[12] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom/agaricus-lepiota.data`

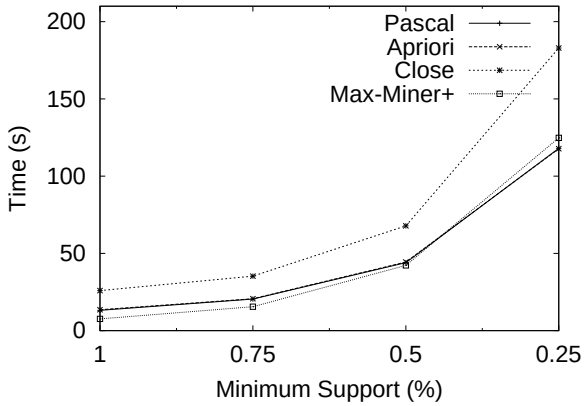

Figure 1: Experimental results for T20I6D100K.

to those obtained for the T20I6D100K dataset: PASCAL and Apriori give identical response times and suffer a slight performance loss over Max-Miner while Close is the worst performer.

In Table 5 and Figure 3, execution times for the T25I10D10K dataset are presented. In this dataset, the proportion of frequent patterns that are not key patterns is much more important than for the T25I20D100K dataset. For the 1.00 and 0.75 minsup thresholds, Max-Miner performs better than Apriori and PASCAL that themselves perform better than Close. For the lower 0.50 and 0.25 minsup thresholds, PASCAL becomes the best performer and is slightly better than Close whereas they both clearly outperform Apriori and Max-Miner: When the proportion of frequent patterns that are not key is significant, the mechanism used by PASCAL (resp. Close) to consider only key (resp. closed) patterns enables to reduce considerably the number of support counts performed.

5.2 Correlated data

Response times obtained from the C20D10K and C73D10K census datasets are given numerically in Tables 6 and 7, and graphically in Figures 4 and 5. Results for the MUSHROOMS dataset are presented in Table 8 and Figure 6. In these three datasets, constituted of correlated data, the proportion of patterns that are frequent is important but few of them are also key patterns. Hence, using pattern counting inference, PASCAL has to perform much fewer support counts than the Apriori and the Max-Miner algorithms. The same observation stands for the Close algorithm, that uses the closure mechanism to reduce the number of support counts, and both PASCAL and Close are an order of magnitude faster than Apriori and Max-Miner. Differences between the execution times of PASCAL and Close and those of Apriori and Max-Miner can be counted in tens of minutes for C20D10K and MUSHROOMS and in hours for C73D10K. Moreover, pattern counting inference and closure mechanism allow to reduce the number of passes on the datasets since the supports of all candidate patterns of some iteration are all deduced from the supports of key, or closed, patterns of previous iterations. On C73D10K with minsup = 60%, for instance, PASCAL and Close both make 13 passes while the largest frequent patterns are of size 19. For this dataset and this threshold value, frequent patterns could not be derived from the maximal frequent patterns extracted with Max-Miner since we did not implement memory management for this phase and it required in this case more memory space than available.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
1.00	583	5.15	5.76	11.15	1.24	1.3
0.75	1,155	9.73	11.13	35.67	1.99	1.77
0.50	1,279,254	968.64	935.14	2,151.34	24.94	879.85

Table 4: Response times for T25I20D100K.

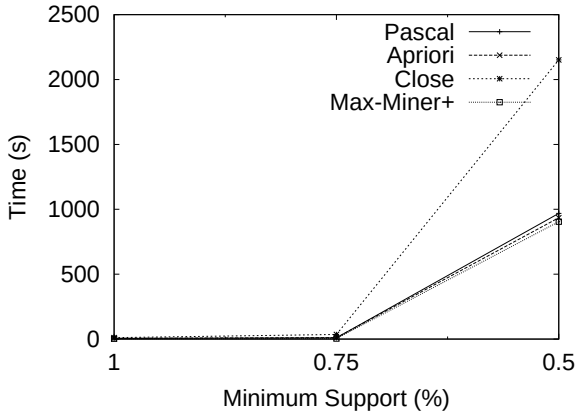

Figure 2: Experimental results for T25I20D10K.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
1.00	3,300	3.24	3.62	6.67	0.63	1.05
0.75	17,583	5.17	6.95	9.38	1.09	3.83
0.50	331,280	17.82	41.06	26.43	2.76	35.53
0.25	2,270,573	70.37	187.92	86.08	6.99	154.89

Table 5: Response times for T25I10D10K.

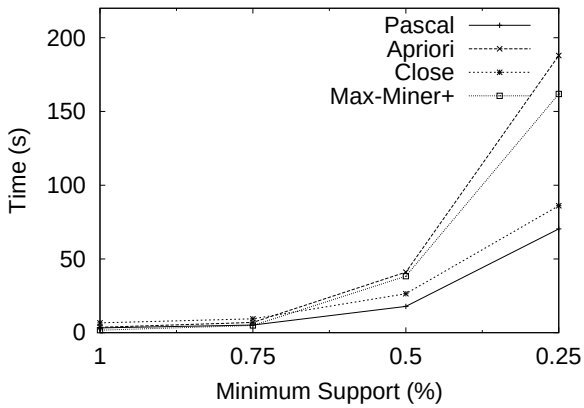

Figure 3: Experimental results for T25I10D10K.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
20.0	20,239	9.44	57.15	14.36	0.17	77.40
15.0	36,359	12.31	85.35	18.99	0.26	113.22
10.0	89,883	19.29	164.81	29.58	0.34	201.33
7.5	153,163	23.53	232.40	36.02	0.35	268.80
5.0	352,611	33.06	395.32	50.46	0.48	428.65
2.5	1,160,363	55.33	754.64	78.63	0.81	775.56

Table 6: Response times for C20D10K.

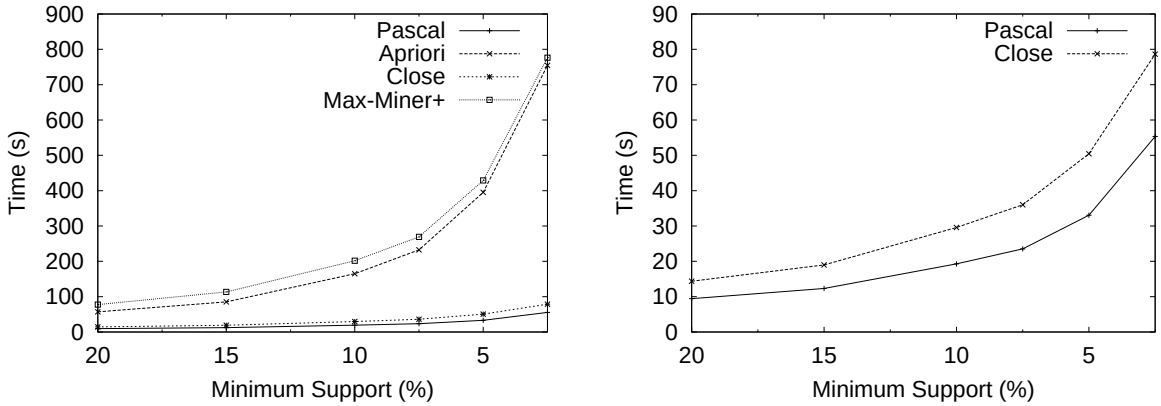

Figure 4: Experimental results for C20D10K.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
80	109,159	177.49	3,661.27	241.91	0.87	3,717.99
75	235,271	392.80	7,653.58	549.27	1.06	7,730.36
70	572,087	786.49	17,465.10	1,112.42	2.28	17,618.40
60	4,355,543	3,972.10	109,204.00	5,604.91	7.72	

Table 7: Response times for C73D10K.

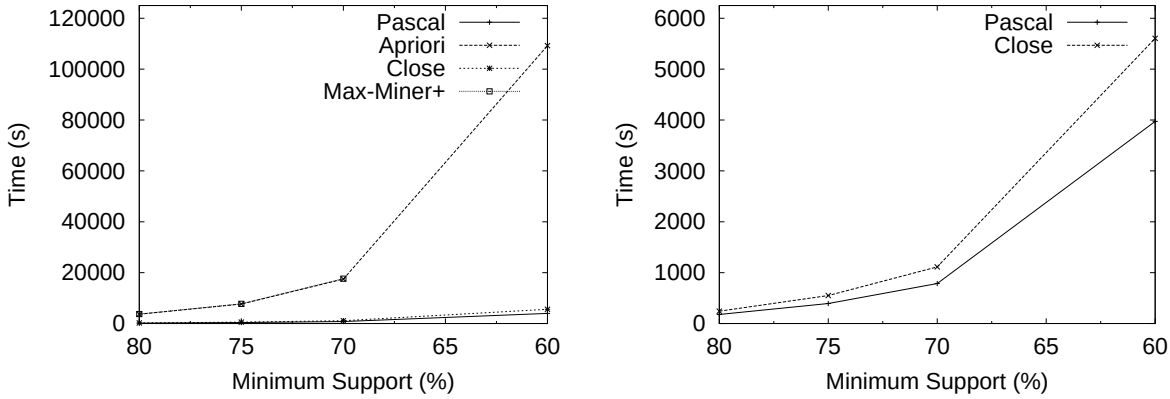

Figure 5: Experimental results for C73D10K.

Support	# frequents	Pascal	Apriori	Close	Max-Miner$^+$	
20.0	53,337	6.48	115.82	9.63	0.31	134.31
15.0	99,079	9.81	190.94	14.57	0.50	218.93
10.0	600,817	23.12	724.35	29.83	0.89	745.72
7.5	936,247	32.08	1,023.24	41.05	1.25	1,035.48
5.0	4,140,453	97.12	2,763.42	98.81	1.99	2,752.05

Table 8: Response times for MUSHROOMS.

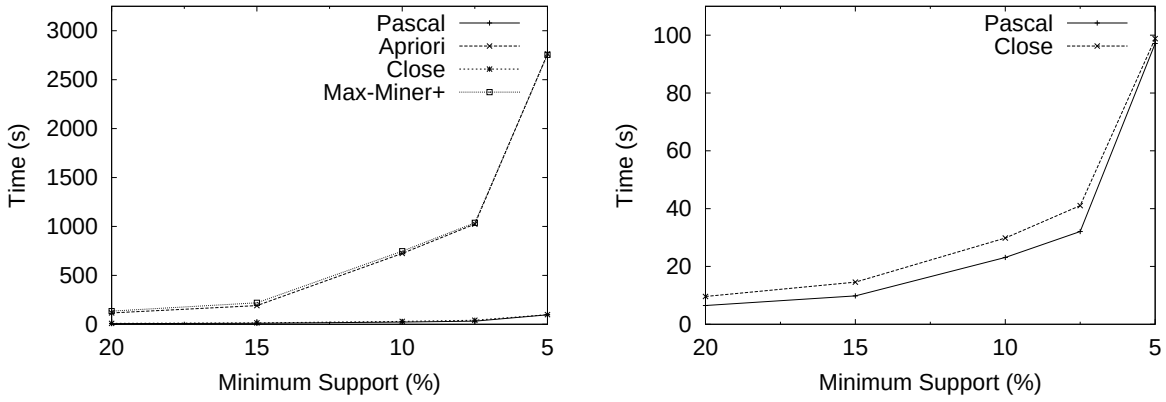

Figure 6: Experimental results for MUSHROOMS.

6 Conclusion

We presented a new algorithm, called PASCAL, for efficiently extracting frequent patterns in large databases. This algorithm is a novel, effective and simple optimization of the Apriori algorithm, thus easy to implement or to integrate into an existing implementation based on the Apriori approach. This optimization, called pattern counting inference, is based on the notion of key patterns of equivalence classes of patterns. It allows to count from the dataset the support of some frequent patterns only, the frequent key patterns, rather than counting the support of all frequent patterns as in algorithms based on the levelwise extraction of frequent patterns or on the extraction of maximal frequent patterns. We conducted performance evaluations to compare the efficiency of PASCAL with those of optimized versions of Apriori, Max-Miner and Close, each one representative of an approach for extracting frequent patterns. The results showed that PASCAL gives response times equivalent to those of Apriori and Max-Miner when extracting all frequent patterns and their support from weakly correlated, and that it is the most efficient among the four algorithms when data are correlated.

We think that an important perspective is the integration of pattern counting inference in Database Management Systems. The integration of data mining methods in relational and object database systems is an important research topic [STA98]. Implementing the PASCAL algorithm in SQL or OQL, we can benefit from database indexing and query processing capabilities, parallelization of the process (e. g., in a SMP environment) and using facilities for checkpointing and space management offered by DBMS for instance.

References

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proc. SIGMOD conf.*, pp 207-216, May 1993.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *Proc. VLDB conf.*, pp 478-499, September 1994.

[AS95] R. Agrawal, and R. Srikant. Mining sequential patterns. *Proc. ICDE conf.*, pp 3-14, March 1995.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. *Proc. SIGMOD conf.*, pp 85-93, June 1998.

[Bay99] R. J. Bayardo and R. Agrawal. Mining the most interesting rules. *Proc. ACM SIGKDD conf.*, pp 145-154, 1999.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *Proc. SIGMOD conf.*, pp 255-264, May 1997.

[BMS97] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlation. *Proc. SIGMOD conf.*, pp 265-276, May 1997.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical foundations.* Springer, 1999.

[GPW98] G. Gardarin, P. Pucheral, and F. Wu. Bitmap based algorithms for mining association rules, *Proc. BDA conf.*, pp 157-175, October 1998.

[HPY00] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation. *Proc. SIGMOD conf.*, pp 1-12, May 2000.

[KHC97] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. *Proc. KDD conf.*, pp 207-210, August 1997.

[LSW97] B. Lent, A. Swami, and J. Widom. Clustering association rules. *Proc. ICDE conf.*, pp 220-231, March 1997.

[LK98] D. Lin and Z. M. Kedem. Pincer-Search: A new algorithm for discovering the maximum frequent set. *Proc. EBDT conf.*, pp 105-119, March 1998.

[MTV94] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. *Proc. AAAI KDD workshop*, pp 181-192, July 1994.

[MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259-289, September 1997.

[MT97] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241-258, September 1997.

[PCY95] J. S. Park, M.-S. Chen, and P. S. Yu, An efficient hash based algorithm for mining association rules. *Proc. SIGMOD conf.*, pp 175-186, May 1995.

[PBTL98] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Pruning closed itemset lattices for association rules. *Proc. BDA conf.*, pp 177-196, October 1998.

[PBTL99a] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25-46, March 1999.

[PBTL99b] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Mining frequent closed itemsets for association rules. *Proc. ICDT Conf.*, pp 398-416, January 1999.

[Pei00] J. Pei, J. Han, and R. Mao. Closet: an efficient algorithm for mining frequent closed itemsets. *Proc. ACM SIGMOD DMKD'00 Workshop*, pp 21-30, May 2000.

[STA98] S. Sarawagi, S. Thomas, R. Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. *Proc. SIGMOD conf.*, pp 343-354, May 1998.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. *Proc. VLDB conf.*, pp 432-444, September 1995.

[SBM98] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39-68, January 1998.

[TPBL00] R. Taouil, N. Pasquier, Y. Bastide, and L. Lakhal. Mining bases for association rules using closed sets. *Proc. ICDE conf.*, poster, p 307, March 2000.

[Toi96] H. Toivonen. Sampling large databases for association rules, *Proc. VLDB conf.*, pp 134-145, September 1996.

[UCI99] S. D. Bay. The UCI KDD Archive [`http://kdd.ics.uci.edu`]. Irvine, CA: University of California, Department of Information and Computer Science.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, New algorithms for fast discovery of association rules. *Proc. KDD conf.*, pp 283-286, August 1997.