Using Condensed Representations for Interactive Association Rule Mining

Baptiste Jeudy and Jean-François Boulicaut[⋆]

Institut National des Sciences Appliquées de Lyon
Laboratoire d'Ingénierie des Systèmes d'Information
Bâtiment Blaise Pascal
F-69621 Villeurbanne cedex, France
Baptiste.Jeudy,Jean-Francois.Boulicaut@lisi.insa-lyon.fr

Abstract. Association rule mining is a popular data mining task. It has an interactive and iterative nature, i.e., the user has to refine his mining queries until he is satisfied with the discovered patterns. To support such an interactive process, we propose to optimize sequences of queries by means of a cache that stores information from previous queries. Unlike related works, we use condensed representations like free and closed itemsets for both data mining and caching. This results in a much more efficient mining technique in highly correlated data and a much smaller cache than in previous approaches.
Keywords association rule, inductive databases, knowledge cache.

1 Introduction

An important data mining problem is the extraction of association rules [1]. It can be stated as follows in the context of basket analysis. The input is a transactional database where each row describes a transaction, i.e., a basket of items bought together by customers. If X is an itemset, i.e., a set of items, its frequency in the database, denoted as $\text{Freq}(X)$, is the number of rows/transactions where all items of X are true/present. An association rule is a pattern $X \Rightarrow Y$ where X and Y are itemsets, its frequency is the frequency of $X \cup Y$ and its confidence measures the conditional probability that a customer buys items from Y knowing that he bought items from X. The standard association rule mining problem concerns the computation of the frequency and confidence of all the association rules that satisfy user-defined constraints such as syntactical constraints, frequency and/or confidence constraints. These latter constraints are based on the so-called objective interestingness measures and specify that the measure (frequency, confidence) must be greater or equal to a user-defined threshold.

From the user point of view, association rule mining is an interactive and iterative process. The user defines a query by specifying various constraints on the rules he wants, e.g., the confidence must be more than 95% or the occurrence

[⋆] 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.

of some items is mandatory. However, when a discovery process starts, it is difficult to figure out the collection of constraints that leads to an interesting result. The result of a data mining query is often unpredictable and the users have to produce sequences of queries until he gets an actionable collection of rules.

Computing the answer of a single association rule query is quite expensive and has motivated many research the last 5 years. It is known that the most expensive step for the standard association rule mining task is the computation of the frequent itemsets, or more generally the computation of the frequency of the interesting itemsets from which the rules will be derived. Indeed, deriving the rules needs for the frequencies of the itemsets to be able to compute the objective interestingness measures like confidence without any access to the raw data. "Pushing" the user-defined constraints can speed up this computation [16, 18]. However, in highly correlated data, there are too many frequent itemsets and the task might be intractable. In this case, the use of condensed representations w.r.t. frequency queries [11] is useful. Several researchers have studied the efficient computation of condensed representations of (frequent) itemsets like the closed sets [17, 4, 20], the free sets [5] or the disjunct free sets [7].

Given a set $\mathcal{S}$ of pairs $(X, \text{Freq}(X))$, we consider that a condensed representation of $\mathcal{S}$ is a subset of $\mathcal{S}$ with two properties: (1) It is much smaller than $\mathcal{S}$ and faster to compute, and (2), the whole set $\mathcal{S}$ can be generated from the condensed representation with no access to the database, i.e., very efficiently. User-defined constraints can also be used to further optimize the computation of condensed representations [6, 10].

However these techniques optimize only one single query. To support the optimization of sequences of queries, we can make use of the "similarities" between these queries that are often refinements of previous queries. It motivates the design of algorithms that try to use the results of already computed queries.

Contribution. In this paper, we propose an algorithm to mine interactively closed itemsets. The user defines constraints on the closed sets and can refine them in a sequence of queries. Our algorithm uses free sets as a cache to store information from the evaluation of previous queries. By using closed sets, our algorithm can be used in highly correlated data where approaches based on itemsets are not usable. Also, our cache of free itemsets is much smaller than a cache containing itemsets and our algorithm ensures that the intersection between the union of the results of all previous queries and the result of the new query is not recomputed. Finally, we do not make any assumption on the relation between two queries in the sequence, e.g., we do not require that the answer of one query is included in the answer of another. In our experiments, we show that this algorithm actually improves the performance of the extraction w.r.t. an algorithm that mines the closed sets without making use of the previous computations. The speedup is roughly equal to the relative size of the intersection between the answer to a new query and the content of the cache. We also show that the size of our cache is always smaller than a cache with itemsets and several orders of magnitude smaller in highly correlated data.

Related Work. Optimizing sequences of data mining queries by caching techniques has been studied for sequences [19] or association rules [9, 2, 15, 8]. However none of these works makes use of condensed representations and the problem of the size of the stored information is not studied. In [15], experiments with different cache sizes are performed but no solution is given in the case of highly correlated data, i.e., when the number of frequent itemsets explodes. Also, most of these works require that some strong relation holds between the queries like inclusion or equivalence. In [14], a more general problem is addressed. A query consists of the constraints defining the subset of a relational database to be mined and the constraints on the association rules themselves. Any of these two sets of queries can be changed by the user. The authors show how to combine algorithms for incremental mining (i.e., when the database changes) with algorithms that only consider changes in the constraints on the association rules. As a result, it is not necessary to consider simultaneously changes on the mined database and on the association rule constraints. In this paper, we focus on changes on the constraints for the closed sets and we assume that the database does not change.

In Sect. 2, we provide some preliminary definitions and introduce the condensed representations based on closed sets and free sets. We propose in Sect. 3 an algorithm that computes constrained closed itemsets without a cache. It is extended in Sect. 4 to use a cache of free itemsets. Finally, we provide an experimental validation in Sect. 5 and Sect. 6 is a short conclusion.

2 Preliminary Definitions

Assume that `Items` is a finite set of symbols denoted by capital letters, e.g., `Items`$= \{$`A`$,$`B`$,$`C`$,\ldots\}$. A *transactional database* is a collection of rows where each row is a subset of `Items`. An *itemset* is a subset of `Items`. A row r *supports* an itemset S if $S \subseteq r$. The *support* (denoted support(S)) of an itemset S is the multi-set of all rows of the database that support S. The *frequency* of an itemset S is the cardinality of support(S) and is denoted Freq(S). Figure 1 provides an example of a transactional database and the supports and the frequencies of some itemsets. We use a string notation for itemsets, e.g., `AB` for $\{$`A`$,$`B`$\}$.

$Db =$

r_1	ABCDE
r_2	AB
r_3	ABD
r_4	CD

Itemset	Support	Frequency
A	$\{r_1, r_2, r_3\}$	3
D	$\{r_1, r_3, r_4\}$	3
AD	$\{r_1, r_3\}$	2
ABCDE	$\{r_1\}$	1

Fig. 1. A four rows transactional database and some itemsets

Definition 1 (Query). *A constraint is a predicate from the power set of* `Items` *to* $\{$*true, false*$\}$. *A query is a pair* $(\mathcal{C}, Db)$ *where* Db *is a transactional database and* $\mathcal{C}$ *is a constraint. The result of a query* $Q = (\mathcal{C}, Db)$ *is defined as the set* $\mathrm{SAT}(Q) = \{(S, \mathit{Freq}(S)), \mathcal{C}(S) = \mathit{true}\}$.

A particular constraint is the minimal frequency constraint $\mathcal{C}_{\gamma-\mathrm{freq}}$ when a frequency threshold γ is given: $\mathcal{C}_{\gamma-\mathrm{freq}}(S) \equiv (\mathrm{Freq}(S) \geq \gamma)$.

Example 1. Given the database *Db* defined in Fig. 1, let us consider the queries $Q_1 = (\mathcal{C}_{2-\mathrm{freq}}, Db)$ and $Q_2 = (\mathcal{C}_{2-\mathrm{freq}} \wedge \mathcal{C}_{miss}, Db)$ where $\mathcal{C}_{miss}(S) \equiv (\mathtt{B} \notin S)$. The answers to these queries are: $\mathrm{SAT}(Q_1) = \{(\emptyset, 4), (\mathtt{A}, 3), (\mathtt{B}, 3), (\mathtt{C}, 2), (\mathtt{D}, 3), (\mathtt{AB}, 3), (\mathtt{AD}, 2), (\mathtt{BD}, 2), (\mathtt{CD}, 2), (\mathtt{ABD}, 2)\}$ and $\mathrm{SAT}(Q_2) = \{(\emptyset, 4), (\mathtt{A}, 3), (\mathtt{C}, 2), (\mathtt{D}, 3), (\mathtt{AD}, 2), (\mathtt{CD}, 2)\}$.

A classical result is that effective safe pruning can be achieved when considering anti-monotone constraints [12, 16].

Definition 2 (Anti-monotonicity). *An* anti-monotone *constraint is a constraint $\mathcal{C}$ such that for all itemsets S, S': $(S' \subseteq S \wedge \mathcal{C}(S)) \Rightarrow \mathcal{C}(S')$.*

The prototypical anti-monotone constraint is the frequency constraint. The constraint $\mathcal{C}_{miss}$ of Example 1 is another anti-monotone constraint and many other examples can be found, e.g., in [16]. Notice that the conjunction or the disjunction of anti-monotone constraints is anti-monotone.

2.1 Closed and Free Itemsets

The concept of closed set is classical within lattice theory and has been studied for association rule mining since the definition of the Close algorithm in [17]. The collection of (frequent) closed itemsets is a useful condensed representation of the (frequent) itemsets in the case of highly correlated data [4].

Definition 3 (closure, closed itemset). *The closure, denoted $cl(S)$, of an itemset S is the largest superset of S that has the same frequency than S. A closed itemset is an itemset S such that $S = cl(S)$.*

The closure operator has some useful properties that are straightforwardly derived from the definition.

Proposition 1.
- $S \subseteq cl(S)$.
- $cl(cl(S)) = cl(S)$.
- *if* $S \subseteq T$ *then* $cl(S) \subseteq cl(T)$.
- $Freq(S) = Freq(cl(S))$.

The second item of this Prop. 1 shows that the closure of an itemset is a closed itemset.

Definition 4 (inherited closure). *The inherited closure of an itemset S is*

$$i\_cl(S) = \bigcup_{T \subset S,\ |T| = |S| - 1} cl(T) \setminus S.$$

The third item of Prop. 1 shows that the inherited closure of an itemset S is actually included in the closure of S. It follows from the first item of Prop. 1 that the disjoint union of S and its inherited closure is included in its closure. We can now define the proper closure of an itemset.

Definition 5 (proper closure). *The proper closure of an itemset S is:*

$$p\_cl(S) = cl(S) \setminus (i\_cl(S) \cup S).$$

The following proposition holds.

Proposition 2. *Let S be an itemset, the closure of S is the disjoint union of S, $i\_cl(S)$ and $p\_cl(S)$.*

Definition 6 (free itemset). *An itemset S is free if it is not included in the closure of any of its proper subsets.*

Indeed, it is sufficient to consider only the proper subsets of size $|S| - 1$. Let us illustrate these definitions on an example.

Example 2. Given the database of Fig. 1, cl($\mathtt{A}$) = $\mathtt{AB}$ and cl($\mathtt{C}$) = $\mathtt{CD}$. Therefore $\mathtt{AC}$ is free and i_cl($\mathtt{AC}$) = $\mathtt{BD}$. p_cl($\mathtt{AC}$) = $\mathtt{E}$ and cl($\mathtt{AC}$) = $\mathtt{ABCDE}$. The closed itemsets are $\emptyset$, $\mathtt{D}$, $\mathtt{AB}$, $\mathtt{CD}$, $\mathtt{ABD}$ and $\mathtt{ABCDE}$. A condensed representation using closed sets of the answer of the query Q_1 of Example 1 is $\{(\emptyset, 4), (\mathtt{D}, 3), (\mathtt{AB}, 3), (\mathtt{CD}, 2), (\mathtt{ABD}, 2)\}$.

There is a strong relationship between closed and free itemsets: the set of closed itemsets is exactly the set of the closures of the free itemsets. This property is used in the following algorithms: To compute the closed sets, the algorithms mine the free itemsets and then output their closures. Let us note that free sets are special cases of δ-free sets [5] and have been independently formalized as key patterns in [3].

3 Mining Constrained Closed Itemsets

We propose an algorithm to mine closed itemsets under any anti-monotone constraint. This algorithm is an extension of the Close algorithm described in [17] in which only the frequency constraint is considered.

Each itemset S is stored in a record also denoted S with four fields: $S.items$ is the list of the items in S, $S.i\_cl$ is the inherited closure, $S.p\_cl$ is the proper closure and $S.freq$ is the frequency of S. In the algorithm, we also use the macro $S.cl$ to denote $S.items \cup S.i\_cl \cup S.p\_cl$.

This is a level-wise algorithm: itemsets of size 0 are considered in the first iteration, then those of size 1, ... At each iteration, the set of candidate itemsets (Cand) is filtered to remove those that do not satisfy the constraint $\mathcal{C}_{am}$ (Step 3). Then a scan on the transactional database is performed to compute the proper closure and the frequency of each itemset in Cand. The candidate itemsets that are not frequent are removed (Step 5) and the closure of the frequent ones are output (Step 6). Then, the candidates for the next iteration are computed using the procedure cand_gen. As in the Apriori algorithm [1], a candidate is generated by joining two itemsets of size k that share the same $k-1$ first items in lexicographic order (e.g., joining `ABC` and `ABD` produces `ABCD`). This procedure also initializes the inherited closure of each new candidate itemset according to Definition 4. Finally, the new candidate itemsets that are not free are removed (Step 8).

Algorithm 1
`Input:` A query $Q = (\mathcal{C}_{\gamma-\text{freq}} \wedge \mathcal{C}_{am}, Db)$ where $\mathcal{C}_{am}$ is an anti-monotone constraint.
`Output:` $\mathcal{O} = \{(\text{cl}(S), \text{Freq}(S)),\ S \text{ is free and } \mathcal{C}_{\gamma-\text{freq}}(S) \wedge \mathcal{C}_{am}(S) \text{ is true}\}$. By construction, $\mathcal{O}$ is a condensed representation of SAT(Q)

```
1  Cand := {(∅, ∅, ∅, 0)}
2  while Cand ≠ ∅ do
3      Cand := {$S$ ∈ Cand, $\mathcal{C}_{am}(S.items)$ = true}
4      DB_pass(Cand, $Db$)
5      Cand := {$S$ ∈ Cand, $S.freq \geq \gamma$}
6      Output({($S.cl$, $S.freq$), $S$ ∈ Cand})
7      Cand := cand_gen(Cand)
8      Cand := {$S$ ∈ Cand, $S$ is free}
9  od
```

An advantage of this algorithm is that it makes an active use of the anti-monotone constraint $\mathcal{C}_{am}$ to prune the search space (and not only the frequency constraint). In previous papers [6, 10], we studied how to push monotone constraints (a monotone constraint is the negation of an anti-monotone one). However, dealing with sequences of queries when monotone constraints are pushed is still under progress.

Algorithm 1 computes the free itemsets that satisfy the constraint $\mathcal{C}_{\gamma-\text{freq}} \wedge \mathcal{C}_{am}$ and then output their closures. It is therefore possible that some of these closures do not satisfy the constraint $\mathcal{C}_{am}$ (but they satisfy $\mathcal{C}_{\gamma-\text{freq}}$ by the fourth item of Prop. 1).

Let us now give an algorithm that generates SAT(Q) from this condensed representation.

Regeneration algorithm
`Input:` The output $\mathcal{O}$ of Alg. 1 and the constraint $\mathcal{C}_{am}$.
`Output:` The answer SAT(Q) to query $Q = (\mathcal{C}_{\gamma-\text{freq}} \wedge \mathcal{C}_{am}, Db)$.

```
1  $\mathcal{I} := \{(S, \text{Freq}(S)),\ \exists C \in \mathcal{O} \text{ s.t. } S \subseteq C\}$
2  Output({$(S, \text{Freq}(S))$, $S \in \mathcal{I}$ and $\mathcal{C}_{am}(S)$ = true})
```

Details about Step 1 can be found in previous works on closed itemsets (see, e.g., [17]) and are not provided here.

Once we know SAT(Q), it is possible to derive association rules by testing (w.r.t. minimal confidence) the rules that can be built from the subsets of each set S from the answer SAT(Q). Notice that an alternative would be to generate non-redundant association rules directly from the output of Alg. 1 [20].

In this process, the expensive part is Alg. 1 and we now discuss its optimization. Indeed, the computation of the answer to query Q by the regeneration algorithm and the generation of association rules do not require further access to the database such that they can be performed efficiently.

4 Caching Free Itemsets

Let us consider a new algorithm that stores information from previous extractions in a *cache* to speed up new extractions using different constraints. First, we describe the structure of this cache and its contents.

In Alg. 1 Line 4, the proper closure and the frequency of each candidate free itemset is computed during a database scan. If this information has been already computed for a previous query, it would be interesting to store it and reuse it. Therefore, we use a cache of free itemsets S with the information computed during the database scan, i.e. p_cl(S) and Freq(S) [1]. The cache is simply a set of records of the form ($S.items, S.p\_cl, S.freq$).

We require that the cache is downward closed. It means that if a free itemset S is in the cache, then every subset of S that is free is also in the cache. This guarantees that if a free itemset is not in the cache then none of its super-sets is in the cache. This property is used to speed up the search of an itemset in the cache.

This cache has been implemented using a prefix tree to store the free itemsets. With this structure, the complexity of the search of an itemset in the cache is proportional to the size of the itemset and not to the size of the cache.

4.1 Algorithm 2

The proof of the completeness and soundness of Alg. 2 is not provided due to the lack of space. However, it might appear clear for a reader familiar with the use of level-wise algorithms that compute closed itemsets.

A new boolean field $S.in\_cache$ is added to the record representing each itemset. This field is used to know if the itemset is in the cache.

The difference between Alg. 1 and Alg. 2 is that the latter uses a cache. During Step 5, the flag $S.in\_cache$ is checked for every itemset in Cand. If it is false, then the itemset cannot be in the cache. If it is true, then the itemset is searched in the cache. If it is found, $S.freq$ and $S.p\_cl$ are updated, else $S.in\_cache$ is set to false. In Step 6, the frequency and proper closure of the itemsets that were not found in the cache (i.e., $S.in\_cache = false$) are computed during a database

[1] It is possible to put closed sets in the cache but some computations would be needed to generate p_cl(S) and Freq(S)

scan and inserted in the new cache (Step 9). During the candidate generation (Step 10), the field *S.in_cache* of every new candidate is initialized with the conjunction of *T.in_cache* for every subset T of S such that $|T| = |S| - 1$. This ensures that this field is false if and only if a subset of S is not in the cache. Since the cache is downward closed, this would mean that S cannot be in the cache.

Algorithm 2
`Input`: A query $Q = (\mathcal{C}_{\gamma-\mathrm{freq}} \wedge \mathcal{C}_{am}, Db)$ where $\mathcal{C}_{am}$ is an anti-monotone constraint and a cache C.
`Output`: The collection $\{(\mathrm{cl}(S), \mathrm{Freq}(S)),\ S$ is free and $\mathcal{C}_{am}(S)$ is true.$\}$ and a new cache C_{new}.

```
1   Cand := $\{(\emptyset, \emptyset, \emptyset, 0, true)\}$
2   $C_{new} := C$
3   **while** Cand $\neq \emptyset$ **do**
4       Cand := $\{S \in$ Cand, $\mathcal{C}_{am}(S.items) =$ true$\}$
5       Cache_pass(Cand, $C$)
6       DB_pass2(Cand, $Db$)
7       Insert_in_Cache(Cand, $C_{new}$)
8       Cand := $\{S \in$ Cand, $S.freq \geq \gamma\}$
9       Output($\{(S.cl, S.freq),\ S \in$ Cand$\}$)
10      Cand := cand_gen2(Cand)
11      Cand := $\{S \in$ Cand, $S$ is free$\}$
12  **od**
```

Algorithm 2 does not make any assumption on the content of the cache except that it is downward closed. This means that it can deals with sequences of queries where no strict relation of inclusion holds between the queries, e.g., it can use results from the computation of the query $Q_3 = (\mathcal{C}_{0.1-\mathrm{freq}}, Db)$ to speed up the computation of $Q_4 = (\mathcal{C}_{0.05-\mathrm{freq}} \wedge \mathcal{C}, Db)$ where $\mathcal{C}(S) \equiv (S \cap \mathtt{AB} = \emptyset)$ even though there is no containment relation between the results of these two queries.

We can formally characterize the content of the cache. Assuming that we already performed the extractions for queries $Q_1 = (\mathcal{C}_1, Db)$, $Q_2 = (\mathcal{C}_2, Db)$, $\ldots$, $Q_n = (\mathcal{C}_n, Db)$, then the cache stores information on the frequency and the proper closures of all free itemsets manipulated during these extractions.

In [12], it is shown that in the case of Apriori, the set of itemsets whose frequency is computed is the set of frequent itemsets plus its negative border, i.e., the set of minimal (w.r.t. the set inclusion) infrequent itemsets. This can be generalized in our framework.

Proposition 3. *Assuming that we already performed the extractions for queries $Q_1 = (\mathcal{C}_1, Db)$, $Q_2 = (\mathcal{C}_2, Db)$, $\ldots$, $Q_n = (\mathcal{C}_n, Db)$, the cache contains the frequencies and proper closures of the free itemsets of* $\mathrm{SAT}(Q_1) \cup \mathrm{SAT}(Q_2) \ldots \cup \mathrm{SAT}(Q_n)$ *plus the minimal free itemsets that do not satisfy $\mathcal{C}_1 \wedge \mathcal{C}_2 \ldots \wedge \mathcal{C}_n$.*

This property describes which information is stored in the cache at a given point. However, this property is not necessary for the completeness or the soundness of Alg. 2 (see next subsection).

4.2 Caching Strategies

In [15], several caching strategies are presented. The strategy that we use is similar to the No Replacement (NR) strategy of [15] (except that our cache is stored in a prefix tree), i.e. itemsets are added in the cache and never removed. This is motivated by the fact that our cache of free itemsets is much smaller than a cache of itemsets and therefore less likely to become full (see Sect. 5).

In the following, we discuss how to adapt the other strategies from [15] to our cache.

The Simple Replacement (SR) strategy uses the fact that it is more valuable to store in the cache the itemsets with the largest frequency because they are more likely to be used in subsequent queries. Thus, when the cache is full, the itemsets with the smallest frequency are removed to store new itemsets. This strategy is easily adaptable to our framework. Removing the free itemsets with the smallest frequency from our cache does not break the downward closure property (because the frequency is a decreasing function w.r.t. the set inclusion). Of course, in this case, Prop. 3 no longer holds.

The Benefit Replacement (BR) strategy from [15] was pointed out as the most efficient. The authors propose to store in the cache a *gsup* value for every k such that every itemset of size k whose frequency is above *gsup* is guaranteed to be in the cache. This can dramatically improve the performance if the new query has a frequency threshold above *gsup*: the algorithm just has to scan the cache to answer the query (thus saving the candidate generation steps). The main problem of this strategy is to compute *gsup*. If queries with only a frequency constraints are used, it is straightforward. With more complex queries [15] gives no solution.

However, it is possible to extend this BR strategy. Let $Q_1 = (\mathcal{C}_1, Db)$, ..., $Q_n = (\mathcal{C}_n, Db)$ be a sequence of queries and $Q = (\mathcal{C}, Db)$ be the new query. If the implication $\mathcal{C}_1 \wedge \ldots \mathcal{C}_n \Rightarrow \mathcal{C}$ holds, then it means that the answer to query Q is in the cache and it is possible to answer it by scanning the cache once.

This strategy shows that it would be quite valuable to combine caching techniques with algorithms that can find such implications in the queries.

5 Experimentations

In this section, we use a relative frequency instead of an absolute frequency: the relative frequency is the absolute frequency divided by the number of rows in the database.

The algorithms have been implemented in Ocaml[2]. All the experiments were conducted on a PC under Linux operating system with an AMD Duron 700Mhz

[2] Developed at INRIA http://caml.inria.fr/ocaml/index.html

processor and 384Mb of memory. We used two common datasets for our experiments, `connect-4` and `mushroom` from the QUEST project[3]. The main particularity of these data sets is that they are dense and highly correlated.

In the first experiment, we study the efficiency when using the cache. When half of the data needed by the algorithm is stored in the cache, we can expect that the computation time is half of the computation without a cache. To evaluate this efficiency, we performed an extraction to build a cache C with a query Q. Then we considered n queries $Q_1, Q_2, \ldots, Q_n$. For each query Q_i, we performed two extractions, one using the cache C (duration dc_i) and another with no cache (duration dnc_i). We denote t_i the total number of candidates that are searched in the cache during Step 5 of Alg. 2 and f_i the number of them that are in the cache. We define the speedup as $1-dc_i/dnc_i$ and the hit rate as $1-f_i/t_i$. Figure 2 represents the hit rate versus the speedup for the two data sets `mushroom` and `connect-4`. It shows that the cache is used efficiently and that using a cache can bring about a significant speedup. In the `mushroom` data set, we can even notice that the speedup is above the hit rate for low hit rates.

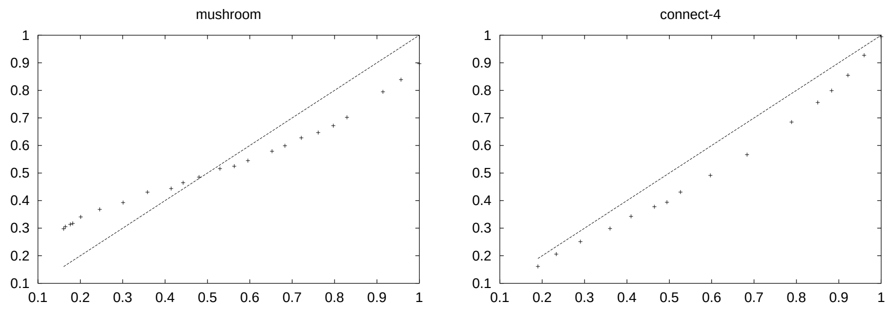

Fig. 2. The speedup versus the hit rate for the `mushroom` data set (left) and the `connect-4` data set (right)

Next, we made several tests to verify our claim about the fact that the complexity of the search in the cache does not depend on the size of the cache. For this, we built two caches C_1 and C_2, C_1 was twenty times larger than C_2. Then we performed two extractions with a query Q such that itemsets used by this query were present either in both caches or in none of them. The first extraction was made using cache C_1 and the second with C_2. There was no significant differences between the two extractions, showing that the size of the cache has no significant impact on the performance.

Finally, we compared the size of our cache of free itemsets with the knowledge cache of [15] that uses frequent itemsets (strategy NR). For each free itemset S in our cache, we count one unit of storage for each item in S plus one unit for

[3] http://www.almaden.ibm.com/cs/quest/

each item in i_cl(S) plus one unit to store the frequency, thus the total size of our cache is $\sum_{S\in C}(|S| + |\text{i\_cl}(S)| + 1)$. For a cache C' of "classical" itemsets, we count one unit for each item of each itemset plus one unit for the frequency, the total size is therefore $\sum_{S\in C'}(|S| + 1)$. With these definitions, we can prove that our cache is *always* smaller than a cache using itemsets. Figure 3 shows that in practical experiments, the actual difference is up to several orders of magnitude.

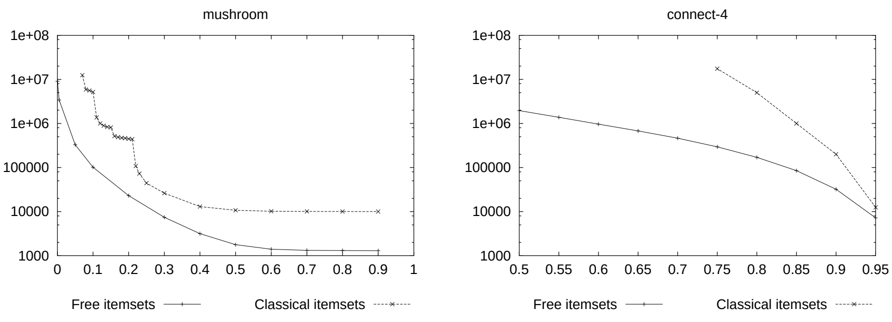

Fig. 3. Size of our cache (free sets) and the itemset cache versus the frequency threshold in the `mushroom` dataset (left) and the `connect-4` data set (right)

6 Conclusion

In this work, we extended the Close algorithm to deal efficiently with a sequence of queries using anti-monotone constraints. To achieve this, we demonstrate the added-value of condensed representations as a knowledge cache for interactive association rule mining.

This work has two major advantages versus previous works on using caches like [15]. First, the use of condensed representations allows mining in highly correlated data where other techniques are not tractable. Second, using these condensed representations leads to a cache that is orders of magnitude smaller that a traditional cache of frequent itemsets.

This cache enables an efficient evaluation of sequences of association rule mining queries and such a technique might be implemented, e.g., within the `MINE RULE` operator [13]. Another perspective of this work is to consider conjunctions of anti-monotone and monotone constraints and study in depth the optimizations in that wider framework.

References

1. Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA, 1996.

2. Elena Baralis and Giuseppe Psaila. Incremental refinement of mining queries. In *Proc. DaWaK'99*, volume 1676 of *LNCS*, pages 173–182. Springer-Verlag, 1999.
3. Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, December 2000.
4. Jean-François Boulicaut and Artur Bykowski. Frequent closures as a concise representation for binary data mining. In *Proc. PAKDD'00*, volume 1805 of *LNAI*, pages 62–73, Kyoto, JP, April 2000. Springer-Verlag.
5. Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by means of free-sets. In *Proc. PKDD'00*, volume 1910 of *LNAI*, pages 75–85, Lyon, F, September 2000. Springer-Verlag.
6. Jean-François Boulicaut and Baptiste Jeudy. Mining free-sets under constraints. In *Proc. IDEAS'01*, pages 322–329, Grenoble, F, July 2001. IEEE Computer Society.
7. Artur Bykowski and Christophe Rigotti. A condensed representation to find frequent patterns. In *Proc. PODS'01*, pages 267–273, Santa Barbara, California, USA, May 2001. ACM Press.
8. Cheikh T. Diop, Arnaud Giacometti, Dominique Laurent, and Nicolas Spyratos. Composition of mining contexts for efficient extraction of association rules. In *Proc. EDBT'02*, Praha, CZ, March 2002. Springer-Verlag. To appear.
9. Bart Goethals and Jan van den Bussche. On implementing interactive association rule mining. In *Proc. DMKD'99*, Philadelphia, USA, May 1999.
10. Baptiste Jeudy and Jean-François Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis, IOS Press*, 6(5), 2002. To appear.
11. Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations. In *Proc. SIGKDD'96*, pages 189–194, Portland, USA, August 1996. AAAI Press.
12. Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
13. Rosa Meo, Giuseppe Psaila, and Stefano Ceri. An extension of SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.
14. Tadeusz Morzy, Marek Wojciechowski, and Maciej Zakrzewicz. Materialized data mining views. In *Proc. PKDD'00*, volume 1910 of *LNAI*, pages 65–74, Lyon, F, September 2000. Springer-Verlag.
15. Biswadeep Nag, Prasad M. Deshpande, and David J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In *Proc. SIGKDD'99*, pages 244–253. ACM Press, 1999.
16. Raymond Ng, Laks V.S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. SIGMOD'98*, pages 13–24, Seattle, Washington, USA, 1998. ACM Press.
17. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, January 1999.
18. Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In *Proc. SIGKDD'97*, pages 67–73, Newport Beach, California, USA, 1997. AAAI Press.
19. Marek Wojciechowski. Interactive constraint-based sequencial pattern mining. In *Proc. ADBIS'01*, volume 2151 of *LNCS*, pages 169–181, Vilnius, Lithuania, September 2001. Springer-Verlag.
20. Mohammed Javeed Zaki. Generating non-redundant association rules. In *Proc. SIGKDD'00*, pages 34–43, Boston, USA, August 2000. AAAI Press.