# On Efficiency of Dataset Filtering Implementations in Constraint-Based Discovery of Frequent Itemsets

Marek WOJCIECHOWSKI, Maciej ZAKRZEWICZ
*Poznan University of Technology, Institute of Computing Science,*
*ul. Piotrowo 3a, 60-965 Poznan, Poland*
*{marek,mzakrz}@cs.put.poznan.pl*

**Abstract**. Discovery of frequent itemsets is one of the fundamental data mining problems. Typically, the goal is to discover all the itemsets whose support in the source dataset exceeds a user-specified threshold. However, very often users want to restrict the set of frequent itemsets to be discovered by adding extra constraints on size and contents of the itemsets. Many constraint-based frequent itemset discovery techniques have been proposed recently. One of the techniques, called dataset filtering, is based on the observation that for some classes of constraints, itemsets satisfying them can only be supported by transactions that satisfy the same constraints. Conceptually, dataset filtering transforms a given data mining task into an equivalent one operating on a smaller dataset. In this paper we discuss possible implementations of dataset filtering, evaluating their strengths and weaknesses.

## 1. Introduction

Discovery of frequent itemsets is one of the fundamental data mining problems. Informally, frequent itemsets are subsets frequently occurring in a collection of sets of items. Discovery of frequent itemsets is a key step in association rule mining [1] but the itemsets themselves also provide useful information on the correlations between items in the database. Typically, the goal is to discover all the itemsets whose support in the source dataset exceeds a user-specified threshold.

The most popular algorithm performing the above task is Apriori introduced in [3]. Apriori reduces the search space by exploiting the following property: an itemset cannot be frequent if any of its subsets is not frequent. The algorithm iteratively generates candidate itemsets from previously found smaller frequent itemsets, and then verifies them in the database. Apriori can be regarded as a classic algorithm that served as a basis for many Apriori-like algorithms offering various performance improvements in frequent itemset mining or adapted to discovery of other types of frequent patterns. Recently, a new family of pattern discovery algorithms, called pattern-growth methods [5], has been developed for discovery of frequent itemsets and other patterns. The methods project databases based on the currently discovered frequent patterns and grow such patterns to longer ones in corresponding projected databases. Pattern-growth methods are supposed to perform better than Apriori-like algorithms in case of low minimum support thresholds. Nevertheless, practical studies [10] show that for real datasets Apriori (or its variants) might still be a better solution.

It has been observed that very often users want to restrict the set of frequent itemsets to be discovered by adding extra constraints on size and contents of the itemsets. It is obvious that additional constraints for itemsets can be verified in a post-processing step, after all itemsets exceeding a given minimum support threshold have been discovered. Nevertheless, such a solution cannot be considered satisfactory since users providing advanced selection criteria may expect that the data mining system will exploit them in the mining process to improve performance. In other words, the system should concentrate on itemsets that are interesting from the user's point of view, rather than waste time on discovering itemsets the user has not asked for [4].

Many constraint-based frequent itemset discovery techniques have been proposed recently for various constraint models. One of the techniques, called dataset filtering [9], is based on the observation that for some classes of constraints, itemsets satisfying them can only be supported by transactions that satisfy the same constraints. The method is distinct from other constraint-based discovery techniques which modify the candidate generation procedure of Apriori [6][8]. Dataset filtering can be applied to two simple but useful in practice types of constraints: the minimum required size of an itemset and the presence of a given subset of items in the itemset.

Conceptually, dataset filtering transforms a given data mining task into an equivalent one operating on a smaller dataset. Thus, it can be integrated with other constraint-based pattern discovery techniques within any pattern discovery algorithm. In this paper we focus on the integration of dataset filtering techniques with the classic Apriori algorithm for the discovery of frequent itemsets. We discuss possible implementations of dataset filtering within Apriori, evaluating their strengths and weaknesses.

### *1.1 Related Work*

Item constraints in frequent itemset (and association rule) mining were first discussed in [8]. Constraints considered there had a form of a Boolean expression in the disjunctive normal form built from elementary predicates requiring that a certain item is or is not present. The algorithms presented were Apriori variants using sophisticated candidate generation techniques.

In [6], two interesting classes of itemset constraints were introduced: anti-monotonicity and succinctness, and methods of handling constraints belonging to these classes within the Apriori framework were presented. The methods for succinct constraints again consisted in modifying the candidate generation procedure. For anti-monotone constraints it was observed that in fact almost no changes to Apriori are required to handle them. A constraint is anti-monotone if the fact that an itemset satisfies it, implies that all of its subsets have to satisfy the constraint too. The minimum support threshold is an example of an anti-monotone constraint, and any extra constraints of that class can be used together with it in candidate pruning.

In [7], constraint-based discovery of frequent itemsets was analyzed in the context of pattern-growth methodology. In the paper, further classes of constraints were introduced, some of which could not be incorporated into the Apriori framework.

### *1.2 Organization of the Paper*

In section 2 we provide basic definitions concerning discovery of frequent itemsets and review of the classic Apriori algorithm. Section 3 presents possible implementations of

dataset filtering techniques within Apriori. Section 4 presents and discusses the results of the experiments that we conducted to evaluate and compare the performance gains offerred by particular implementations of dataset filtering. We conlude with a summary of achieved results in section 5.

## 2. Background

### *2.1 Basic Definitions*

Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. An *itemset* $X$ is a non-empty set of items ($X \subseteq L$). The *size* of an itemset $X$ (denoted as $|X|$) is the number of items in $X$. Let $D$ be a set of variable size itemsets, where each itemset $T$ in $D$ has a unique identifier and is called a *transaction*. We say that a transaction $T$ *supports* an item $x \in L$ if $x$ is in $T$. We say that a transaction $T$ *supports* an itemset $X \subseteq L$ if $T$ supports every item in the set $X$. The *support* of the itemset $X$ is the percentage of transactions in $D$ that support $X$. The problem of mining frequent itemsets in $D$ consists in discovering all itemsets whose support is above a user-defined support threshold.

Given two itemsets $X$ and $Y$ such that $Y \subseteq X$, $X' = X \setminus Y$ (the set difference of $X$ and $Y$) is called a *projection* of $X$ with respect to the subset $Y$. Given a database $D$ and an itemset $Y$, a *$Y$-projected database* can be constructed from $D$ by removing transactions that do not support $Y$, and then replacing the remaining transactions by their projections with respect to $Y$.

### *2.2 Review of the Apriori Algorithm*

Apriori relies on the property that an itemset can only be frequent if all of its subsets are frequent. It leads to a level-wise procedure. First, all possible 1-itemsets (itemsets containing 1 item) are counted in the database to determine frequent 1-itemsets. Then, frequent 1-itemsets are combined to form potentially frequent 2-itemsets, called candidate 2-itemsets. Candidate 2-itemsets are counted in the database to determine frequent 2-itemsets. The procedure is continued by combining the frequent 2-itemsets to form candidate 3-itemsets and so forth. The algorithm stops when in a certain iteration none of the candidates turns out to be frequent or the set of generated candidates is empty.

## 3. Dataset Filtering Within the Apriori Framework

In our simple constraint model we assume that a user specifies two extra constraints together with the minimum support threshold: the required subset $S$ and the minimum size threshold $s$. Thus, the problem consists in discovering all frequent itemsets including $S$ and having size greater than $s$. We assume that a user may specify both extra constraints or only one of them. If the latter is the case, $S = \emptyset$ or $s = 0$, depending on which constraint has been omitted. The constraints we consider are simple examples of item and size constraints. However, it should be noted that these constraints are more difficult to handle within the Apriori framework than their negations: the requirement that a certain set is not included in an itemset and the maximum allowed size of an itemset. The latter two constraints are anti-monotone and therefore can be handled by Apriori in the same way the minimum support constraint is used.

On the other hand, the constraints we consider can be handled by dataset filtering techniques. It is obvious that itemsets including the set $S$ can be supported only by

transactions also supporting *S*, and itemsets whose size exceeds *s* can be supported only by transactions of size greater than *s*. Therefore, according to the idea of dataset filtering, the actual frequent itemset discovery can be performed on the subset on the source dataset consisting from all the transactions supporting *S*, and having size greater than *s*. It should be noted that frequent itemsets discovered in the filtered dataset may also include those not supporting the item or size constraints (the post-processing itemset filtering phase is still required).

The correctness of application of dataset filtering for our constraint model comes from the fact that the number of transactions supporting itemsets satisfying the constraints will be the same in the original and filtered datasets. It should be noted that the support of itemsets not satisfying user-specified constraints, counted in the filtered dataset, can be smaller than their actual support in the original dataset, but it is not a problem since these itemsets will not be returned to the user. Moreover, this is in fact a positive feature as it can reduce the number of generated candidates not leading to itemsets of user's interest. Since we assume that a user specifies the minimum support threshold as a percentage of the total number of transactions in the source dataset, in all implementations of Apriori with dataset filtering, during the first iteration the required number of supporting transactions is derived from the support threshold provided by a user and the total number of transactions found in the dataset.

Regarding integration of dataset filtering with the Apriori algorithm, there are two general implementation strategies. The filtered dataset can either be physically materialized on disk during the first iteration or filtering can be performed on-line in each iteration. We also observe that if the required subset is not empty, the idea of projection with respect to the required subset can by applied to reduce the size of the filtered dataset (meaningful if the filtered dataset is to be materialized) and the number of iterations. In such a case, frequent itemsets are being discovered in the projected dataset and then are extended with the required subset with respect to which the projection has been performed. If apart from the required subset constraint, the minimum size threshold is also present, projection has to be coupled with filtering according to the size constraint. Thus, we have four possible implementations of dataset filtering within the Apriori framework, leading to the following four Apriori variants (all the algorithms presented below take a collection *D* of transactions, the minimum support threshold, the required subset *S*, and the minimum size threshold *s* as input, and return all frequent itemsets in *D* satisfying all the provided constraints).

**Algorithm 1** (Apriori on materialized filtered dataset)

```
begin
  scan D in order to:
    1) evaluate minimum number of supporting
       transactions for an itemset to be called frequent (mincount)
    2) find L1 (set of 1-itemsets supported by at
       least mincount transactions supporting S and having size > s);
    3) materialize the collection D' of transactions from D
       supporting S and having size > s;
  for (k = 2; Lk-1 ≠ ∅; k++) do
  begin
    /* generate new candidates using a standard Apriori procedure */
    Ck = apriori_gen(Lk-1);
    if Ck = ∅ then break;
    forall transactions d ∈ D' do
      forall candidates c ∈ Ck do
        if d supports c then
          c.count ++;
        end if;
```

```
    Lk = { c ∈ Ck | c.count ≥ mincount};
  end;
  output itemsets from ∪k Lk having S as subset and size > s;
end.
```

**Algorithm 2** (Apriori with on-line dataset filtering)

```
begin
  scan D in order to:
    1) evaluate minimum number of supporting
       transactions for an itemset to be called frequent (mincount)
    2) find L1 (set of 1-itemsets supported by at
       least mincount transactions supporting S and having size > s);
  for (k = 2; Lk-1 ≠ ∅; k++) do
  begin
    /* generate new candidates using a standard Apriori procedure */
    Ck = apriori_gen(Lk-1);
    if Ck = ∅ then break;
    forall transactions d ∈ D do
      if d supports S and |d|>s then
        forall candidates c ∈ Ck do
          if d supports c then
            c.count ++;
          end if;
      end if;
    Lk = { c ∈ Ck | c.count ≥ mincount};
  end;
  output itemsets from ∪k Lk having S as subset and size > s;
end.
```

**Algorithm 3** (Apriori on materialized projected dataset)

```
begin
  scan D in order to:
    1) evaluate minimum number of supporting
       transactions for an itemset to be called frequent (mincount)
    2) find L1 (set of 1-itemsets supported by at
       least mincount transactions from S-projected dataset D' of
       transactions from D supporting S and having size > s);
    3) materialize the S-projected dataset D' of transactions from D
       supporting S and having size > s;
  for (k = 2; Lk-1 ≠ ∅; k++) do
  begin
    /* generate new candidates using a standard Apriori procedure */
    Ck = apriori_gen(Lk-1);
    if Ck = ∅ then break;
    forall transactions d ∈ D' do
      forall candidates c ∈ Ck do
        if d supports c then
          c.count ++;
        end if;
    Lk = { c ∈ Ck | c.count ≥ mincount};
  end;
  forall itemsets X ∈ ∪k Lk do
    if |X∪S| > s then output X∪S;
    end if;
  end;
end.
```

**Algorithm 4** (Apriori with on-line dataset projection)

```
begin
  scan D in order to:
    1) evaluate minimum number of supporting
       transactions for an itemset to be called frequent (mincount)
    2) find L1 (set of 1-itemsets supported by at
       least mincount transactions from S-projected dataset D' of
       transactions from D supporting S and having size > s);
  for (k = 2; Lk-1 ≠ ∅; k++) do
  begin
    /* generate new candidates using a standard Apriori procedure */
    Ck = apriori_gen(Lk-1);
    if Ck = ∅ then break;
    forall transactions d ∈ D do
      if d supports S and |d|>s then
        forall candidates c ∈ Ck do
          if d supports c then
            c.count ++;
          end if;
      end if;
    Lk = { c ∈ Ck | c.count ≥ mincount};
  end;
  forall itemsets X ∈ ∪k Lk do
    if |X∪S| > s then output X∪S;
    end if;
  end;
end.
```

## 4. Experimental Results

In order to compare performance gains offered by various implementations of dataset filtering, we performed several experiments on a PC with the *Intel Celeron* 266MHz processor and 96 MB of main memory. The experiments were conducted on a synthetic dataset generated by means of the *GEN* generator from the *Quest* project [2], using different item and size constraints, and different values of the minimum support threshold. In each experiment, we compared execution times of applicable implementations of Apriori with dataset filtering and the original Apriori algorithm extended with the post-processing itemset filtering phase.

The source dataset was generated using the following values of *GEN* parameters: total number of transactions = 10000, the number of different items = 1000, the average number of items in a transaction = 8, number of patterns = 500, average pattern length = 3. The generated dataset was stored in a flat file on disk.

We started the experiments with varying item and size constraints for the fixed minimum support threshold of 1.5%. Apart from measuring the total execution times of all applicable Apriori implementations, we also registered the selectivity of dataset filtering constraints (expressed as the percentage of transactions in the database satisfying dataset filtering constraints). Figures 1 and 2 present execution times of various implementations of extended Apriori for size and item constraints of different selectivity (PP – original Apriori with a post-processing filtering phase, OL – on-line dataset filtering, MA – dataset filtering with materialization of the filtered dataset, POL – on-line dataset projection, PMA - projection with materialization of the projected dataset).

As we expected, the experiments showed that the lower the selectivity of dataset filtering constraints, the better the performance gains due to dataset filtering or projection are likely to be as compared to the original Apriori. It is obvious that the selectivity of a particular dataset

filtering constraint depends on the actual contents of the database. In general, we observed that item constraints led to much better results (reducing the processing time 2 to 8 times depending on constraint selectivity and filtering implementation method) than constraints referring only to itemset size (typically reducing the processing time by less than 10%). This is due to the fact that frequent itemsets to be discovered are usually smaller than transactions forming the source dataset, and therefore even restrictive size constraints on frequent itemsets result in weak constraints on transactions.

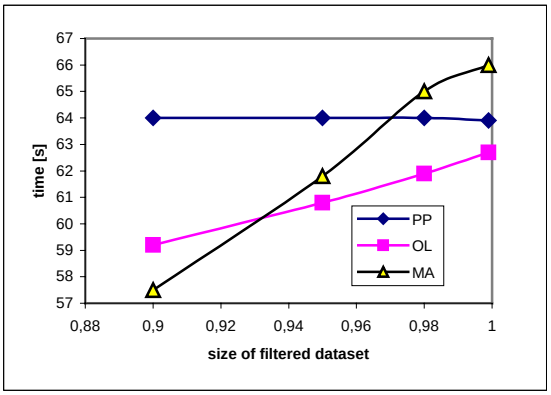


Fig. 1. Execution times for different values of selectivity of size constraints

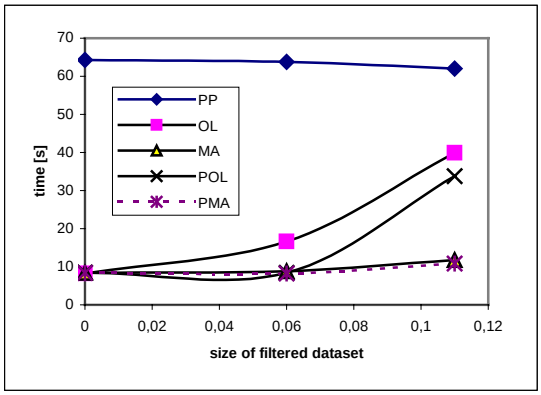


Fig. 2. Execution times for different values of selectivity of item constraints

In case of item constraints, all the implementations of dataset filtering and projection were always more efficient than the original Apriori with a post-processing constraint verifying step. Projection led to better results than filtering, which can be explained by the fact that projection leads to the smaller number of Apriori iterations (and slightly reduces the size of transactions in the dataset). Implementations involving materialization of the filtered/projected dataset were more efficient than their on-line counterparts (the filtered/projected dataset was relatively small and the materialization cost was dominated by gains due to the smaller costs of dataset scans in candidate verification phases). However, in case of size constraints rejecting a very small number of transactions, materialization of the filtered dataset sometimes lead to longer execution times than in case of the original Apriori. The on-line dataset filtering implementation was in general more efficient than the original Apriori even for size constraints (except for a situation, unlikely in practice, when the size constraint did not reject any transactions).

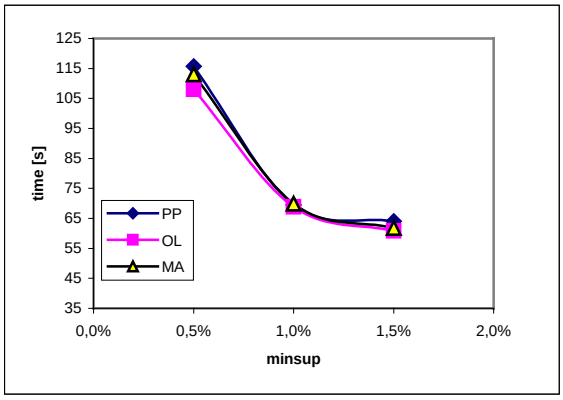


Fig. 3. Execution times for different values of minimum support in presence of size constraints

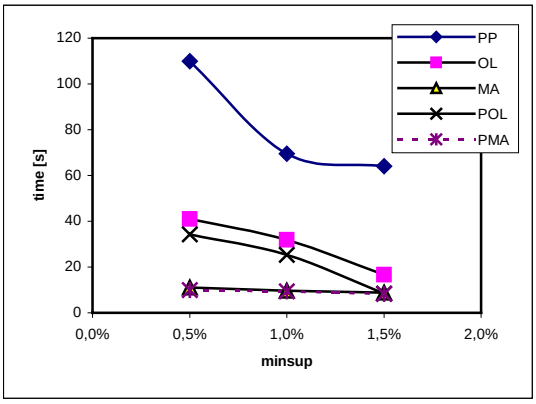


Fig. 4. Execution times for different values of minimum support in presence of item constraints

In another series of experiments, we observed the influence of varying the minimum support threshold on performance gains offered by dataset filtering and projection. Figure 3 presents the execution times for a size constraint of selectivity 95%. Execution times for an item constraint of selectivity 6% are presented in Figure 4. In both cases the minimum support

threshold varied from 0.5% to 1.5%. Apriori encounters problems when the minimum support threshold is low because of the huge number of candidates to be verified. In our experiments, decreasing the minimum support threshold worked in favor of dataset filtering techniques, especially in case of item constraints leading to a small filtered dataset. This behavior can be explained by the fact that since dataset filtering reduces the cost of candidate verification phase, the more this phase contributes to the overall processing time, the more significant relative performance gains are going to be (the lower the support threshold the more candidates to verify, while the cost of disk access remains the same). Decreasing the minimum support threshold also led to slight performance improvement of implementations involving materialization of the filtered/projected dataset in comparison to their on-line counterparts. As the support threshold decreases, the maximal length of a frequent itemsets (and the number of iterations required by the algorithms) increases. Materialization is performed in the first iteration and reduces the cost of the second and subsequent iterations. Thus, the more iterations are required, the better the cost of materialization is compensated.

## 5. Conclusions

In this paper we addressed the issue of frequent itemset discovery with item and size constraints. One possible method of handling such constraints is application of dataset filtering techniques which are based on the observation that for certain types of constraints, some of the transactions in the database can be excluded from the discovery process since they cannot support the itemsets of interest.

We discussed several possible implementations of dataset filtering within the classic Apriori algorithm. Experiments show that dataset filtering can be used to improve performance of the discovery process but the actual gains depend on the type of the constraint and the implementation method. Item constraints typically lead to much more impressive performance gains than size constraints since they result in a smaller size of the filtered dataset. The best implementation strategy for handling item constraints is materialization of the database projected with respect to the required subset, whereas for size constraints the best results should be achieved by on-line filtering of the database with no materialization.

## References


[1] R. Agrawal, T. Imielinski, A. Swami, Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conference on Management of Data, 1993.

[2] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, T. Bollinger, The Quest Data Mining System. Proc. of the 2nd KDD Conference, 1996.

[3] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. Very Large Data Bases, 1994

[4] J. Han, L. Lakshmanan, R. Ng, Constraint-Based Multidimensional Data Mining. *IEEE Computer* **32** (1999)

[5] J. Han and J. Pei, Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. *SIGKDD Explorations*, December 2000 (2000)

[6] R. Ng, L. Lakshmanan, J. Han, A. Pang, Exploratory Mining and Pruning Optimizations of Constrained Association Rules. Proceedings of the 1998 ACM SIGMOD Conference on Management of Data, 1998.

[7] J. Pei, J. Han, L. Lakshmanan, Mining Frequent Itemsets with Convertible Constraints. Proceedings of the 17th ICDE Conference, 2001.

[8] R. Srikant, Q. Vu, R. Agrawal, Mining Association Rules with Item Constraints. Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997.

[9] M. Zakrzewicz, Data Mining within DBMS Functionality. Proceedings of the 4th IEEE International Baltic Workshop on Databases & Information Systems, 2000.

[10] Z. Zheng, R. Kohavi, L. Mason, Real World Performance of Association Rule Algorithms. Proc. of the 7th KDD Conference, 2001.