A Study on Answering a Data Mining Query Using a Materialized View*

Maciej Zakrzewicz, Mikolaj Morzy, Marek Wojciechowski

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

Abstract. One of the classic data mining problems is discovery of frequent itemsets. This problem particularly attracts database community as it resembles traditional database querying. In this paper we consider a data mining system which supports storing of previous query results in the form of materialized data mining views. While numerous works have shown that reusing results of previous frequent itemset queries can significantly improve performance of data mining query processing, a thorough study of possible differences between the current query and a materialized view has not been presented yet. In this paper we classify possible differences into six classes, provide I/O cost analysis for all the classes, and experimentally evaluate the most promising ones.

1 Introduction

Data mining aims at discovery of useful patterns from large databases or warehouses. Nowadays we are witnessing the evolution of data mining environments from specialized tools to multi-purpose data mining systems offering some level of integration with existing database management systems. Data mining can be seen as advanced querying, where a user specifies the source dataset and the requested pattern constraints, then the system chooses the appropriate data mining algorithm and returns the discovered patterns to the user. Data mining query processing has recently become an important research area focusing mainly on constraint handling and reusing results of previous queries.

In our previous work we introduced the concept of materialized data mining views, providing a general discussion on their possible usage in mining various classes of frequent patterns [8][9]. In this paper we focus on the most prominent class of patterns – frequent itemsets. We present a thorough study of possible differences between the current frequent itemset query and a materialized view. We identify six classes of possible differences, providing I/O cost analysis for each of them. For the most promising classes we report results of conducted experiments.

* This work was partially supported by the grant no. 4T11C01923 from the State Committee for Scientific Research (KBN), Poland.

1.1 Background

Frequent itemsets. Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. Let a non-empty set of items T be called an *itemset*. Let D be a set of variable length itemsets, where each itemset $T \subseteq L$. We say that an itemset T *supports* an item $x \in L$ if x is in T. We say that an itemset 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 itemsets 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 *minsup*.

Apriori algorithm. *Apriori* is an example of a level-wise algorithm for frequent itemset discovery. It makes multiple passes over the input data to determine all frequent itemsets. Let L_k denote the set of frequent itemsets of size k and let C_k denote the set of candidate itemsets of size k. Before making the k-th pass, *Apriori* generates C_k using L_{k-1}. Its candidate generation process ensures that all subsets of size $k-1$ of C_k are all members of the set L_{k-1}. In the k-th pass, it then counts the support for all the itemsets in C_k. At the end of the pass all itemsets in C_k with a support greater than or equal to the minimum support form the set of frequent itemsets L_k. Figure 1 provides the pseudocode for the general level-wise algorithm, and its *Apriori* implementation. The *subset(t, k)* function gives all the subsets of size k in the set t.

```
C1 = {all 1-itemsets from D}
for (k=1; Ck ≠ ∅; k++)
        count(Ck, D);
        Lk = {c ∈ Ck | c.count ≥ minsup};
        Ck+1 = generate_candidates(Lk);
Answer = ∪k Lk;
```

```
L1 = {frequent 1-itemsets}
for (k = 2; Lk-1 ≠ ∅; k++)
        Ck = generate_candidates(Lk-1);
        forall tuples t ∈ D
                Ct = Ck ∩ subset(t, k);
                forall candidates c ∈ Ct
                        c.count++;
        Lk = {c ∈ Ck | c.count ≥ minsup}
Answer = ∪k Lk;
```

Fig. 1. A general level-wise algorithm for association discovery (left) and its Apriori implementation (right)

1.2 Related Work

The problem of association rule discovery was introduced in [1]. In the paper, discovery of frequent itemsets was identified as the key step in association rule mining. In [3], the authors proposed an efficient frequent itemset mining algorithm called *Apriori*. Since it has been observed that generation of association rules from frequent itemsets is a straightforward task, further research focused mainly on efficient methods for frequent itemset discovery.

Incremental mining in the context of frequent itemsets and association rules was first discussed in [5]. A novel algorithm called *FUP* was proposed to efficiently discover frequent itemsets in an incremented dataset, exploiting previously discovered frequent itemsets.

The notion of data mining queries (or *KDD* queries) was introduced in [7]. The need for Knowledge and Data Management Systems (KDDMS) as second generation data mining tools was expressed. The ideas of application programming interfaces and data mining query optimizers were also mentioned.

In [10] the authors postulated to create a knowledge cache that would keep recently discovered frequent itemsets along with their support value, in order to facilitate interactive and iterative mining. Besides presenting the notion of knowledge cache the authors introduced several maintenance techniques for such cache, and discussed using the cache contents when answering new frequent set queries.

In [4] three relationships which occur between two association rule queries were identified. The relationships represented cases when results on one query can be used to efficiently answer the other. However, the relationships concerned association rules – not frequent itemsets.

The work on materialized views started in the 80s. The basic concept was to use materialized views as a tool to speed up queries and serve older copies of data. Materialized views have become a key element of data warehousing technology (see [11] for an overview).

2 Basic Definitions and Problem Formulation

Definition 1 (Data mining query). A *data mining query for frequent itemset discovery* is a tuple $dmq=(R, a, \Sigma, \Phi, \beta)$, where R is a database relation, a is a set-valued attribute of R, Σ is a data selection predicate on R, Φ is a selection predicate on frequent itemsets, β is the minimum support for the frequent itemsets. The data mining query *dmq* returns all frequent itemsets discovered in $\pi_a\sigma_\Sigma R$, having support greater than β and satisfying the constraints Φ.

Example. Given is the database relation $R_1(attr_1, attr_2)$. The data mining query $dmq_1 = (R_1, "attr_2", "attr_1>5", "|itemset|<4", 3)$ describes the problem of discovering frequent itemsets in the set-valued attribute $attr_2$ of the relation R_1. The frequent itemsets with support above 3 and length less than 4 are discovered in records having $attr_1>5$.

Definition 2 (Materialized data mining view). A *materialized data mining view* $dmv=(R, a, \Sigma, \Phi, \beta)$ is a data mining query, whose both the definition and the result are permanently stored (materialized) in a database. All frequent itemsets being a result of the data mining query are called *materialized data mining view contents.*

Definition 3 (Restricted frequent itemset selection predicate). Given two data mining queries: $dmq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $dmq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. We say that the frequent itemset selection predicate Φ_1 is *restricted* with respect to the frequent itemset selection predicate Φ_2 (or Φ_2 is *relaxed* with respect to Φ_1), written as $\Phi_2 \subset \Phi_1$, if and only if for each frequent itemset, satisfying Φ_1 implies also satisfying Φ_2. We say that the frequent itemset selection predicates are *independent* if $\Phi_1 \not\subset \Phi_2 \wedge \Phi_2 \not\subset \Phi_1 \wedge \Phi_1 \neq \Phi_2$.

Definition 4 (Stronger frequent itemset selection predicate). Given two selection predicates on frequent itemsets: p_1 and p_2. We say that p_1 is *stronger* than p_2 if any of the conditions shown in Table 1 holds. We assume that items are integers. S represents a frequent itemset, *min*()/*max*() returns the highest/lowest item, *count*() returns the size of an itemset, *sum*() returns the sum of all items, *range*() returns the difference between the highest and the lowest item, V_1 and V_2 are sets of items, v_1 and v_2 are integers.

Table 1. Conditions for p_1 being stronger than p_2

p_1	p_2	condition
$S \supseteq V_1$	$S \supseteq V_2$	$V_1 \supset V_2$
$S \subseteq V_1$	$S \subseteq V_2$	$V_1 \subset V_2$
$min(S) \leq v_1$	$min(S) \leq v_2$	$v_1 < v_2$
$min(S) \geq v_1$	$min(S) \geq v_2$	$v_1 > v_2$
$max(S) \leq v_1$	$max(S) \leq v_2$	$v_1 < v_2$
$max(S) \geq v_1$	$max(S) \geq v_2$	$v_1 > v_2$
$count(S) \leq v_1$	$count(S) \leq v_2$	$v_1 < v_2$
$count(S) \geq v_2$	$count(S) \geq v_2$	$v_1 > v_2$
$sum(S) \leq v_1$ ($\forall x \in S, x \geq 0$)	$sum(S) \leq v_2$ ($\forall x \in S, x \geq 0$)	$v_1 < v_2$
$sum(S) \geq v_1$ ($\forall x \in S, x \geq 0$)	$sum(S) \geq v_2$ ($\forall x \in S, x \geq 0$)	$v_1 > v_2$
$range(S) \leq v_1$	$range(S) \leq v_2$	$v_1 < v_2$
$range(S) \geq v_1$	$range(S) \geq v_2$	$v_1 > v_2$

Theorem 1. Given two data mining queries: $dmq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $dmq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. The frequent itemset selection predicate Φ_1 is *restricted* with respect to the frequent itemset selection predicate Φ_2 if any of the following holds:

(1) The selection predicate Φ_2 is a conjunction of n predicates $p_1^2 \wedge p_2^2 \wedge ... \wedge p_n^2$, the selection predicate Φ_1 is a conjunction of $n+1$ predicates $p_1^1 \wedge p_2^1 \wedge ... \wedge p_n^1 \wedge p_{n+1}^1$, and for each $1 \leq i \leq n$ we have $p_i^1 = p_i^2$.

(2) The selection predicate Φ_1 is a conjunction of n predicates $p_1^1 \wedge p_2^1 \wedge ... \wedge p_{n-1}^1 \wedge p_n^1$, the selection predicate Φ_2 is a conjunction of n predicates $p_1^2 \wedge p_2^2 \wedge ... \wedge p_{n-1}^2 \wedge p_n^2$, for each $1 \leq i \leq (n-1)$ we have $p_i^1 = p_i^2$, and the predicate p_n^1 is stronger than p_n^2.

(3) There exists a frequent itemset selection predicate Φ_3, such that $\Phi_3 \subset \Phi_1 \wedge \Phi_2 \subset \Phi_3$.

Proof. The proof is straightforward, based on definitions 3 and 4.

Definition 5 (Restricted data selection predicate). Given two data mining queries: $dmq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $dmq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. We say that the data selection predicate Σ_1 is *restricted* with respect to Σ_2 (or Σ_2 is *relaxed* with respect to Σ_2), written as $\Sigma_2 \subset \Sigma_1$, if and only if for each record of R, satisfying Σ_1 implies also satisfying Σ_2. We say that the data selection predicates are *independent* if $\Sigma_1 \not\subset \Sigma_2 \wedge \Sigma_2 \not\subset \Sigma_1 \wedge \Sigma_1 \neq \Sigma_2$.

3 Data Mining Query Execution Using a Materialized Data Mining View

Let us consider the problem of executing a data mining query using a materialized data mining view. Let $dmq=(R, a, \Sigma_{dmq}, \Phi_{dmq}, \beta_{dmq})$, $dmv_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$. We will discuss different methods of employing dmv_1 in the process of executing dmq. We enumerate six query-view configuration classes, that enable us to use the materialized data mining view: (1) Class I – identical data selection predicates, identical frequent itemset selection predicates, identical minimum supports, (2) Class II – identical data selection predicates, frequent itemset selection predicate relaxed or independent in *dmq* or minimum support lowered in *dmq*, (3) Class III – identical data selection predicates, frequent itemset selection predicate restricted or equal in *dmq*, minimum support not lowered in *dmq*, (4) Class IV – data selection predicate restricted in dmv_1, identical frequent itemset selection predicates, identical minimum supports, (5) Class V – data selection predicates restricted in dmv_1, frequent itemset selection predicate relaxed or independent in *dmq* or minimum support lowered in *dmq*, (6) Class VI – data selection predicate restricted in dmv_1, frequent itemset selection predicate restricted or equal in *dmq*, minimum support not lowered in *dmq*. Classes I and IV are subclasses of classes III and VI respectively, offering more efficient query answering algorithms. In all other cases (data selection predicates independent or data selection predicate relaxed in *dmq*), dmv_1 is not usable in executing *dmq* (itemsets contained in dmv_1 were counted in parts of database that are not relevant to *dmq*).

Class I ($\Sigma_1=\Sigma_{dmq} \wedge \beta_1=\beta_{dmq} \wedge \Phi_1=\Phi_{dmq}$). Since the materialized data mining view dmv_1 contains the exact result of the data mining query *dmq*, then the execution of *dmq* only takes to read the contents of dmv_1. We will refer to this method as to *View Ready* (*VR*). The I/O cost $cost_{VR}$ for *View Ready* involves only the retrieval of dmv_1 contents:

$$cost_{VR} = ||dmv_1||, \tag{1}$$

where $||dmv_1||$ is the size of dmv_1 contents (disk pages).

In order to estimate the benefits of using *View Ready*, let us consider the I/O cost $cost_{FULL}$ of executing a complete frequent itemset discovery algorithm (eg., *Apriori*) on $\sigma_{\Sigma dmq}R$. The cost involves k scans of $\sigma_{\Sigma dmq}R$ (k depends on the particular algorithm used):

$$cost_{FULL} = k \cdot ||\sigma_{\Sigma dmq}R||, \tag{2}$$

where $||\sigma_{\Sigma dmq}R||$ is the I/O cost of retrieving all records of R satisfying Σ_{dmq}. Notice that *View Ready* is useful if $||dmv_1|| < k \cdot ||\sigma_{\Sigma dmq}R||$. Since in practical application of frequent itemset discovery, we usually have $||dmv_1|| << ||\sigma_{\Sigma dmq}R||$, then it is highly beneficial to use the described method in order to execute a data mining query.

Class II ($\Sigma_1=\Sigma_{dmq} \wedge (\beta_1>\beta_{dmq} \vee \Phi_1 \not\subset \Phi_{dmq})$). Since the materialized data mining view is not guaranteed to contain all itemsets to be returned by *dmq*, the execution of *dmq* takes to perform a simplified frequent itemset discovery algorithm, eg., *Apriori*, in which we count only those candidates, that do not belong to dmv_1. If a candidate belongs to dmv_1, then we do not need to count it, because we already know its support. We will refer to this method as to *Complementary Mining* (*CM*). The I/O cost $cost_{CM}$ for *Complementary Mining* involves *k* scans of $\sigma_{\Sigma dmq}R$ (*k* depends on the particular algorithm used) and a single scan of dmv_1:

$$cost_{CM} = k \cdot ||\sigma_{\Sigma dmq}R|| + ||dmv_1||. \quad (3)$$

When we compare the I/O cost of *Complementary Mining* with the I/O cost of executing a complete frequent itemset discovery algorithm (eg., *Apriori*) on $\sigma_{\Sigma dmq}R$, then we notice that *Complementary Mining* is more costly compared to not using a materialized data mining view at all. This fact actually eliminates *Complementary Mining* from practical applications. However, since the I/O cost is only a part of a total cost of executing a data mining query, then in a very specific case it might happen that the I/O overhead gets compensated by an improvement of CPU time. Such effects may occur e.g., in CPU-bound computer systems.

Class III ($\Sigma_1=\Sigma_{dmq} \wedge \beta_1 \leq \beta_{dmq} \wedge \Phi_1 \subseteq \Phi_{dmq}$). Since the materialized data mining view dmv_1 contains a superset of the result of *dmq*, then the execution of *dmq* takes to read the contents of dmv_1 and filter the frequent itemsets with respect to β_{dmq} and Φ_{dmq}. We will refer to this method as to *Verifying Mining* (*VM*). The I/O cost $cost_{VM}$ for *Verifying Mining* involves only the scan of dmv_1:

$$cost_{VR} = ||dmv_1||. \quad (4)$$

When we compare the I/O cost of *Verifying Mining* with the I/O cost of executing a complete frequent itemset discovery algorithm (e.g., *Apriori*) on $\sigma_{\Sigma dmq}R$, then we notice that *Verifying Mining* is useful if $||dmv_1|| < k \cdot ||\sigma_{\Sigma dmq}R||$. According to our discussion above, we conclude that *Verifying Mining* is highly beneficial.

Class IV ($\Sigma_1 \subset \Sigma_{dmq} \wedge \beta_1=\beta_{dmq} \wedge \Phi_1=\Phi_{dmq}$). The database has been logically divided into two partitions (1) the records covered by the materialized data mining view dmv_1, (2) the records covered by the data mining query *dmq*, and not covered by the materialized data mining view. Since dmv_1 contains frequent itemsets discovered only in the first partition, therefore the executing of *dmq* takes to discover all frequent itemsets in the second partition (eg. using *Apriori*), to merge the discovered frequent itemsets with the frequent itemsets from dmv_1, and finally to scan the database in order to count and filter frequent itemsets. We will refer to this method as to *Incremental Mining* (*IM*) since it is similar to incremental update algorithms. The I/O cost $cost_{IM}$ for *Incremental Mining* involves *k* scans of $\sigma_{(\Sigma dmq-(\Sigma dmq \cap \Sigma 1)}R$ (*k* depends on the particular algorithm used), a sigle scan of dmv_1, and a single scan of $\sigma_{\Sigma dmq}R$:

$$cost_{CM} = k \cdot ||\sigma_{(\Sigma dmq-(\Sigma dmq \cap \Sigma 1)}R|| + ||dmv_1|| + ||\sigma_{\Sigma dmq}R||. \quad (5)$$

When we compare the I/O cost of *Incremental Mining* with the I/O cost of executing a complete frequent itemset discovery algorithm (e.g., *Apriori*) on $\sigma_{\Sigma dmq}R$, then we notice that *Incremental Mining* is useful if: $k\cdot\|\sigma_{(\Sigma dmq-(\Sigma dmq\cap\Sigma 1)}R\| + \|dmv_1\| < (k\text{-}1)\cdot\|\sigma_{\Sigma dmq}R\|$. Assuming that in practical applications we usually have: $\|dmv_1\| << \|\sigma_{(\Sigma dmq-(\Sigma dmq\cap\Sigma 1)}R\| < \|\sigma_{\Sigma dmq}R\|$, it means that *Incremental Mining* is beneficial (in terms of I/O costs) when $\|\sigma_{(\Sigma dmq-(\Sigma dmq\cap\Sigma 1)}R\| < (k\text{-}1)/k\cdot\|\sigma_{\Sigma dmq}R\|$, which means that e.g., for k=10 the materialized data mining view should cover at least 10% of the dataset covered by the data mining query.

Class V ($\Sigma_1 \subset \Sigma_{dmq} \wedge (\beta_1 > \beta_{dmq} \vee \Phi_1 \not\subset \Phi_{dmq})$). The database has been logically divided into two partitions (1) the records covered by the materialized data mining view dmv_1, (2) the records covered by the data mining query *dmq*, and not covered by the materialized data mining view. The materialized data mining view dmv_1 is not guaranteed to contain all the frequent itemsets that would be discovered in the first partition (using β_{dmq} and Φ_{dmq}). The execution of *dmq* is a two-step procedure. In the first step, we execute a simplified frequent itemset discovery algorithm, e.g. *Apriori*, in which we count only those candidates, that do not belong to dmv_1. If a candidate belongs to dmv_1, then we do not need to count it, because we already know its support. In the second step, we discover all frequent itemsets in the second partition, we merge the discovered frequent itemsets with those from the first step, and finally we scan the database to count and filter them. Formally, this method is a combination of *Complementary Mining* and *Incremental Mining*, therefore its I/O cost is the following:

$$cost_{CM} + cost_{IM} = k\cdot\|\sigma_{\Sigma 1}R\| + \|dmv_1\| + k\cdot\|\sigma_{(\Sigma dmq-(\Sigma dmq\cap\Sigma 1)}R\| + \|\sigma_{\Sigma dmq}R\| = (k+1)\ \|\sigma_{\Sigma dmq}R\| + \|dmv_1\|. \quad (6)$$

When we compare the above I/O cost with the I/O cost of executing a complete frequent itemset discovery algorithm on $\sigma_{\Sigma dmq}R$, then we notice that in most practical applications the above method is more costly compared to not using a materialized data mining view at all. However, since the I/O cost is only a part of a total cost of executing a data mining query, then in a very specific case it might happen that the I/O overhead gets compensated by an improvement of CPU time. Such effects may occur e.g., in CPU-bound computer systems.

Class VI ($\Sigma_1 \subset \Sigma_{dmq} \wedge \beta_1 \leq \beta_{dmq} \wedge \Phi_1 \subseteq \Phi_{dmq}$). The database has been logically divided into two partitions (1) the records covered by the materialized data mining view dmv_1, (2) the records covered by the data mining query *dmq*, and not covered by the materialized data mining view. The materialized data mining view dmv_1 contains a superset of all frequent itemsets that would be discovered in the first partition (using β_{dmq} and Φ_{dmq}). The execution of *dmq* is a two-step procedure. In the first step we scan dmv_1 and we filter its frequent itemsets with respect to β_{dmq} and Φ_{dmq}. In the second step, we discover all frequent itemsets in the second partition, we merge the discovered frequent itemsets with those from the first step, and finally we scan the

database to count and filter them. Formally, this method is a combination of *Verifying Mining* and *Incremental Mining*, therefore its I/O cost is the following:

$$cost_{VR} + cost_{IM} = ||dmv_1|| + k \cdot ||\sigma_{(\Sigma dmq - (\Sigma dmq \cap \Sigma 1)} R|| + ||\sigma_{\Sigma dmq} R||. \quad (7)$$

When we compare the above I/O cost with the cost of executing a complete frequent itemset discovery algorithm on $\sigma_{\Sigma dmq} R$, then we notice that the discussed method is useful if: $||dmv_1|| + k \cdot ||\sigma_{(\Sigma dmq - (\Sigma dmq \cap \Sigma 1)} R|| < (k\text{-}1) \cdot ||\sigma_{\Sigma dmq} R||$. Assuming that in most practical applications we have: $||dmv_1|| << ||\sigma_{(\Sigma dmq - (\Sigma dmq \cap \Sigma 1)} R|| < ||\sigma_{\Sigma dmq} R||$, *Verifying Mining* + *Incremental Mining* is beneficial (in terms of I/O costs) if $||\sigma_{(\Sigma dmq - (\Sigma dmq \cap \Sigma 1)} R|| < (k\text{-}1)/k \cdot ||\sigma_{\Sigma dmq} R||$. For instance, for k=10 it means that the materialized data mining view should cover at leas 10% of the dataset covered by the data mining query.

Our above discussion has been summarized in the Table 2.

Table 2. Methods of executing a data mining query using a materialized data mining view

	$\Sigma_1 = \Sigma_{dmq}$	$\Sigma_1 \subset \Sigma_{dmq}$
$\beta_1 = \beta_{dmq} \wedge \Phi_1 = \Phi_{dmq}$	*VR*	*IM*
$\beta_1 > \beta_{dmq} \vee \Phi_1 \subset \Phi_{dmq}$	*CM*	*CM,IM*
$\beta_1 \leq \beta_{dmq} \wedge \Phi_1 \subseteq \Phi_{dmq}$	*VM*	*VM,IM*

4 Experimental Results

In order to evaluate performance gains stemming from using a materialized view, we performed several experiments on a Pentium II 433MHz PC with 128 MB of RAM. We experimented with synthetic and real datasets. The synthetic datasets were generated using the *GEN* generator from the *Quest* project [2]. The real datasets that we used came from the UCI KDD Archive [6]. Here we report results on the MSWeb[1] (Microsoft Anonymous Web Data) dataset and a synthetic dataset containing 148000 transactions built from 100 different items, with the average transaction size of 30.

In the tests we did not consider Class I (trivial, in practice always beneficial) and classes involving *Complementary Mining*, i.e., II and V (theoretically proven as inefficient). Thus, we focused on practical verification of *Verifying Mining* and *Incremental Mining*. As a complete data mining algorithm we used our implementation of *Apriori*. To simulate constraints of a multi-user environment, we limited the amount of main memory available to algorithms to 10-50kB. Each chart presents average results from a series of 20 experiments.

In the first series of experiments we varied the level of coverage of the query's dataset by materialized view's dataset. The minimum support of the query was by 10% higher than in case of the materialized view. Figure 2 presents the results for real and synthetic datasets.

[1] http://kdd.ics.uci.edu/databases/msweb/msweb.html

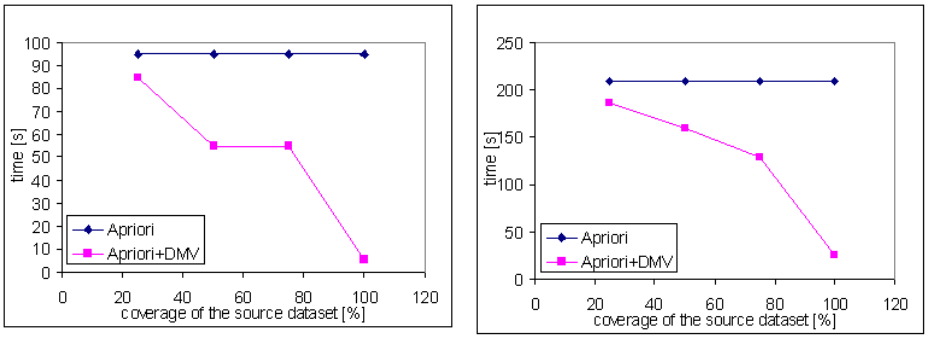

Fig. 2. Execution times for various levels of coverage of the query's dataset by materialized view's dataset for real (left) and synthetic (right) datasets

The experiments show that even for a materialized view based on the dataset covering 20% of the query's dataset, exploiting the results stored in the view reduces processing time. In general, more significant coverage results in better performance of the method using a materialized view. However, the exact performance improvement depends also on data distribution and the support threshold.

In the second series of experiments we tested the impact of difference between the support thresholds of the query to be answered and the materialized data mining view. The results for both considered datasets are presented in Fig. 3. The difference between the thresholds is expressed as the percentage of the support threshold of the query to be answered (the support threshold used for the materialized view was lower than the support threshold used in the query). For both datasets the source dataset for the view covered 75% of the dataset of the query.

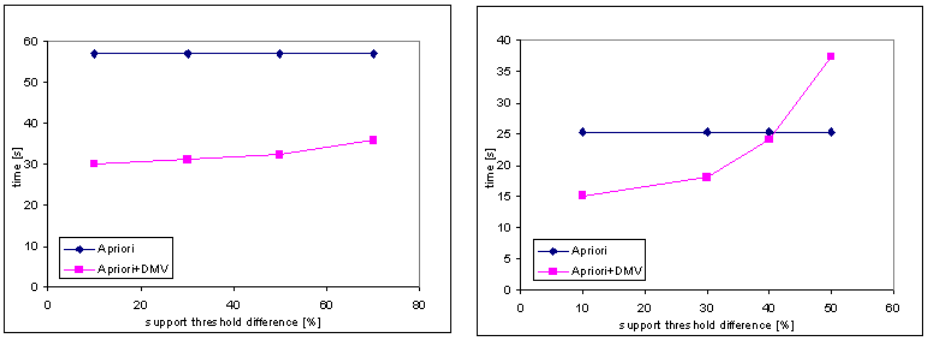

Fig. 3. Execution times for various relative differences in support thresholds for real (left) and synthetic (right) datasets

The experiments show that using a materialized view is more efficient when the difference between the support threshold is small. For big differences it is even possible that using a materialized view is a worse solution than running the complete frequent itemset mining algorithm. This can happen since for a very low support

threshold the size of a materialized view can be very big, introducing high I/O costs. The exact value of the difference between support thresholds for which using a materialized view is not beneficial depends on the actual threshold values, the nature of the dataset, and the coverage of the query's dataset by materialized view's dataset.

5 Conclusions

In this paper we discussed answering a frequent itemset query using a materialized data mining view. We classified possible differences between the current query and the query defining the materialized view into six classes. We provided I/O cost analysis for all the classes, and experimentally evaluated the most promising ones.

Theoretical analysis and experiments show that using a materialized view is an efficient solution in cases for which *View Ready*, *Verifying Mining*, and *Incremental Mining* techniques are applicable. The latter two methods perform particularly well when the support threshold of the view is close to the support threshold of the query and/or the source dataset of the view covers significant part of the query's dataset.

In the future we plan to consider situations in which a given query can be answered using a collection of materialized data mining views.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conf. on Management of Data (1993)
2. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon (1996)
3. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
4. Baralis E., Psaila G.: Incremental Refinement of Mining Queries. Proceedings of the 1st DaWaK Conference (1999)
5. Cheung D.W.-L., Han J., Ng V., Wong C.Y.: Maintenance of discovered association rules in large databases: An incremental updating technique. Proc. of the 12th ICDE Conference (1996)
6. Hettich S., Bay S. D.: The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science (1999)
7. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
8. Morzy T., Wojciechowski M., Zakrzewicz M.: Materialized Data Mining Views. Proceedings of the 4th PKDD Conference (2000)
9. Morzy T., Wojciechowski M., Zakrzewicz M.: Fast Discovery of Sequential Patterns Using Materialized Data Mining Views. Proceedings of the 15th ISCIS Conference (2000)
10. Nag B., Deshpande P.M., DeWitt D.J.: Using a Knowledge Cache for Interactive Discovery of Association Rules. Proceedings of the 5th KDD Conference (1999)
11. Roussopoulos N.: Materialized Views and Data Warehouses. SIGMOD Record, 27(1) (1998)