Efficient Processing of Frequent Itemset Queries Using a Collection of Materialized Views [*]

Marek Wojciechowski and Maciej Zakrzewicz

Poznan University of Technology, ul. Piotrowo 3a, Poznan, Poland

Abstract. One of the classic data mining problems is discovery of frequent itemsets. Frequent itemset discovery tasks can be regarded as advanced database queries specifying the source dataset, the minimum support threshold, and optional constraints on itemsets. We consider a data mining system which supports storing of results of previous queries in the form of materialized data mining views. Previous work on materialized data mining views addressed the issue of reusing results of one of the previous frequent itemset queries to efficiently answer the new query. In this paper we present a new approach to frequent itemset query processing in which a collection of materialized views can be used for that purpose.

1 Introduction

Frequent itemset mining is one of the classic data mining problems, identified as the key step in association rule discovery [1]. Frequent itemsets and association rules capture the co-occurrence of items in the collection of sets, and find numerous applications including market basket analysis and web usage mining. Frequent itemset mining can be seen as advanced querying [6], where a user specifies the source dataset, the minimum support threshold, and optionally some constraints on itemsets, then the system chooses the appropriate data mining algorithm and returns the results 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.

We consider a data mining system which supports storing of results of previous queries in the form of materialized data mining views [7]. In our previous work [10] we addressed the issue of reusing results of one of the previous frequent itemset queries to efficiently answer the new query. In this paper we present a new approach to frequent itemset query processing in which a collection of materialized views can be used for that purpose. We propose a query execution method that uses partial results from a set of materialized views, and provide an algorithm that selects a set of materialized views that is optimal in terms of the I/O cost.

[*] 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* [2] is a classic 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 *minsup* form the set of frequent itemsets L_k.

1.2 Related Work

Incremental mining in the context of frequent itemsets was first discussed in [4]. A novel algorithm called *FUP* was proposed to efficiently discover frequent itemsets in an incremented dataset, exploiting previously discovered frequent itemsets. *FUP* was based on the same generate-and-test paradigm as *Apriori* - it is bound to a particular mining methodology. Using our terminology, *FUP* exploits one materialized view, and cannot be easily extended to use more than one view.

In [8] the authors postulated to create a knowledge cache that would keep recently discovered frequent itemsets. 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 [3] relationships between association rule queries were analyzed. They represented cases when results of 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. Since then, materialized views have become a key element of data warehousing technology (see [9] for an overview).

2 Frequent Itemset Query Execution Using a Single Materialized Data Mining View

In [10] we considered the problem of executing a data mining query using a single materialized data mining view. In this section we review the basic definitions and summarize the results of our previous study.

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

Example. Given is the database relation $\mathcal{R}_1(attr_1, attr_2)$. The data mining query $dmq_1 = (\mathcal{R}_1, "attr_2", "attr_1 > 5", "|itemset| < 4", 3)$ describes the problem of discovering frequent itemsets in the set-valued attribute $attr_2$ of the relation $\mathcal{R}_1$. 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* $dmv = (\mathcal{R}, a, \Sigma, \Phi, \beta)$ is a data mining query, whose both the definition and the result are permanently stored (materialized) in a database. All frequent itemsets being the result of the data mining query are called *materialized data mining view contents.*

For a frequent itemset query $dmq = (\mathcal{R}, a, \Sigma_{dmq}, \Phi_{dmq}, \beta_{dmq})$, and a materialized view $dmv_1 = (\mathcal{R}, a, \Sigma_1, \Phi_1, \beta_1)$, we identified two general cases, presented below, in which dmq can be answered using dmv_1. The cases are described in terms of relationships between selection predicates. $\Sigma_1 \subset \Sigma_{dmq}$ means that the source dataset of dmv_1 is a subset of the source dataset of dmq (e.g., $"attr_1 > 5" \subset "attr_1 > 2"$). $\Phi_1 \subseteq \Phi_{dmq}$ means that if an itemset satisfies Φ_{dmq} then it must also satisfy Φ_1 (e.g., $"|itemset| < 5" \subseteq "|itemset| < 3"$). See [10] for details.

Verifying Mining (VM): $(\Sigma_1 = \Sigma_{dmq} \wedge \beta_1 \leq \beta_{dmq} \wedge \Phi_1 \subseteq \Phi_{dmq})$. 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}. (In a particular case, when $\beta_1 = \beta_{dmq}$ and $\Phi_1 = \Phi_{dmq}$, dmv_1 contains the exact result of dmq, and no filtering is needed.)

Incremental Mining (IM): $(\Sigma_1 \subset \Sigma_{dmq} \wedge \beta_1 \leq \beta_{dmq} \wedge \Phi_1 \subseteq \Phi_{dmq})$. The database has been logically divided into two partitions: (1) the records covered by the view dmv_1, (2) the records covered by the query dmq, and not covered by dmv_1. The execution of dmq consists of three steps. In the first step, the contents of dmv_1 are filtered with respect to β_{dmq} and Φ_{dmq}. (In a particular case, when $\beta_1 = \beta_{dmq}$ and $\Phi_1 = \Phi_{dmq}$ this step is not needed.) In the second step, all itemsets frequent in the second partition are discovered using a complete mining algorithm (e.g., *Apriori*). Finally, locally frequent itemsets from both partitions are merged and then counted during the scan of the database in order to find globally frequent itemsets.

3 Frequent Itemset Query Execution Using a Set of Materialized Data Mining Views

Problem formulation. Given are: (1) a set of materialized data mining views $DMV = \{dmv_1, dmv_2, ..., dmv_n\}$, where $dmv_i = (\mathcal{R}, a, \Sigma_i, \Phi_i, \beta_i)$, Σ_i is of the form $(l^i_{1min} < a < l^i_{1max}) \vee (l^i_{2min} < a < l^i_{2max}) \vee ... \vee (l^i_{kmin} < a < l^i_{kmax})$, $l^i_* \in dom(a)$, and (2) a data mining query $dmq = (\mathcal{R}, a, \Sigma_{dmq}, \Phi_{dmq}, \beta_{dmq})$, where Σ_{dmq} is of the form $(l^{dmq}_{1min} < a < l^{dmq}_{1max}) \vee (l^{dmq}_{2min} < a < l^{dmq}_{2max}) \vee ... \vee (l^{dmq}_{mmin} < a < l^{dmq}_{mmax})$, $l^{dmq}_* \in dom(a)$. The problem of *materialized data mining view selection* consists in generating such an algorithm of dmq execution using views from DMV, that its I/O cost is minimal.

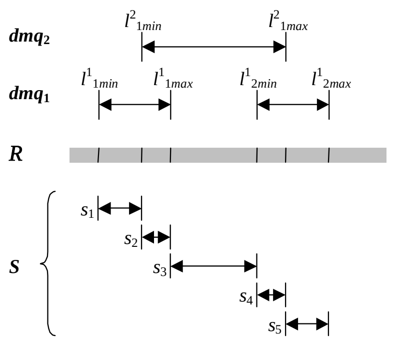

Fig. 1. Sample set of data mining queries and their distinct selection formulas

Definition 3 (Data sharing graph). Let $S = \{s_i, s_2, ..., s_k\}$ be a set of *distinct selection formulas* for $DMV \cup \{dmq\}$, i.e., a set of such selection formulas over the attribute a of $\mathcal{R}$ that for each i, j we have $\sigma_{s_i}\mathcal{R} \cap \sigma_{s_j}\mathcal{R} = \emptyset$, for each i there exist integers $a, b, ..., m$, such that $\sigma_{\Sigma_i}\mathcal{R} = \sigma_{s_a}\mathcal{R} \cup \sigma_{s_b}\mathcal{R} \cup ... \cup \sigma_{s_m}\mathcal{R}$, and there exist integers $a, b, ..., k$, such that $\sigma_{\Sigma_{dmq}}\mathcal{R} = \sigma_{s_a}\mathcal{R} \cup \sigma_{s_b}\mathcal{R} \cup ... \cup \sigma_{s_k}\mathcal{R}$ (Fig. 1). A graph $DSG = (V, E)$ will be called a *data sharing graph* for DMV and a data mining query dmq if and only if $V = DMV \cup S \cup \{dmq\}$,

$E = \{(v, s_j) | v \in DMV \cup \{dmq\},\ s_j \in S,\ \sigma_{\Sigma_v}\mathcal{R} \cap \sigma_{s_j}\mathcal{R} \neq \emptyset\}$.

Example. Let us consider a data sharing graph for a set of materialized data mining views and a data mining query. Given is a relation $\mathcal{R}_1 = (attr_1, attr_2)$, four materialized data mining views: $dmv_1 = (\mathcal{R}_1, "attr_2", "10 < attr_1 < 30", \emptyset, 10)$, $dmv_2 = (\mathcal{R}_1, "attr_2", "15 < attr_1 < 40", \emptyset, 4)$, $dmv_3 = (\mathcal{R}_1, "attr_2", "30 < attr_1 < 40", \emptyset, 8)$, $dmv_4 = (\mathcal{R}_1, "attr_2", "5 < attr_1 < 30", \emptyset, 15)$, and a data mining query $dmq = (\mathcal{R}_1, "attr_2", "5 < attr_1 < 40", \emptyset, 10)$. The set of distinct selection formulas consists of the following elements: $S = \{s_1 = "5 < attr_1 < 10", s_2 = "10 < attr_1 < 15", s_3 = "15 < attr_1 < 30", s_4 = "30 < attr_1 < 40"\}$. The data sharing graph for $DMV = \{dmv_1, dmv_2, dmv_3, dmv_4\}$ and dmq is shown in Fig. 2.

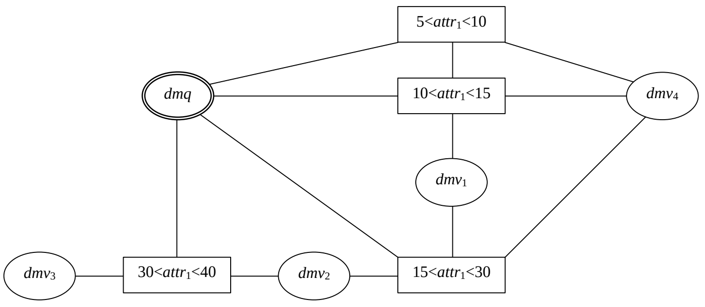

Fig. 2. Sample data sharing graph for a set of materialized data mining views and a data mining query

3.1 Materialized data mining view selection

Consider a data mining query dmq which can be executed using multiple data mining views from DMV. According to our previous analysis, dmq can use such views from DMV that: (1) are based on a subset of dmq's source dataset ($\Sigma_i \subseteq \Sigma_{dmq}$), (2) use *minsup* which is not above the *minsup* of dmq ($\beta_i \leq \beta_{dmq}$), and (3) their data selection conditions are identical to or relaxed with respect to dmq ($\Phi_i \subseteq \Phi_{dmq}$). Moreover, the source datasets of the materialized views must not overlap, since in such case, certain frequent itemsets might be lost if they were not frequent in any of the source datasets. Therefore, in order to efficiently execute the data mining query $dmq = (\mathcal{R}, a, \Sigma_{dmq}, \Phi_{dmq}, \beta_{dmq})$, we should consider the following subsets of materialized views:

$$DMV_{dmq} = \{dmv_i \in DMV | dmv_i = (\mathcal{R}, a, \Sigma_i, \Phi_i, \beta_i), \Sigma_i \subseteq \Sigma_{dmq}, \quad (1)$$
$$\beta_i \leq \beta_{dmq}, \Phi_i \subseteq \Phi_{dmq}, \forall dmv_k \in DMV_{dmq} - \{dmv_i\} : \Sigma_i \cap \Sigma_k = \emptyset\}.$$

Using DMV_{dmq} to execute dmq consists in: (1) selecting frequent itemsets from each $dmv_i \in DMV_{dmq}$ with supports above β_{dmq}, (2) discovering frequent itemsets in the portion of the database, which is not covered by any view from DMV_{dmq}, (3) merging all frequent itemsets from (1) and (2) to form a set of global candidates, and (4) scanning the database to evaluate their support and prune infrequent ones. The algorithm for executing a data mining query dmq using a set DMV_{dmq} of materialized data mining views is shown below.

Algorithm 1 (Data mining query execution using a set of materialized data mining views).
Input: A data mining query dmq and a set of materialized data mining views DMV_{dmq}
Output: The results of dmq.
Method:

```
1.  for each $dmv_i \in DMV_{dmq}$ do
2.     $\mathcal{F}_i \leftarrow \{$frequent itemsets from $dmv_i$ having support $\geq \beta_{dmq}$
              and satisfying $\Phi_{dmq}\}$
3.  $\mathcal{R}' \leftarrow \sigma_{\Sigma_{dmq}}\mathcal{R} - \bigcup_{dmv_i \in DMV_{dmq}} \sigma_{\Sigma_i}\mathcal{R})$
4.  $\mathcal{F}' \leftarrow execute(\mathcal{R}', a, \text{"true"}, \Phi_{dmq}, \beta_{dmq})$
5.  $\mathcal{C} \leftarrow \mathcal{F}' \cup \mathcal{F}_1 \cup \mathcal{F}_2 \cup ... \cup \mathcal{F}_{|DMV_{dmq}|}$
6.  $count(\mathcal{C}, \sigma_{\Sigma_{dmq}}\mathcal{R})$
7.  $Answers \leftarrow \{C \in \mathcal{C} | C.count \geq \beta_{dmq}\}$
8.  return $Answers$
```

Since many applicable subsets of materialized data mining views may exist, we will look for such DMV_{dmq} that the I/O cost of dmq execution is minimal. The I/O cost of executing dmq using DMV_{dmq} is the following:

$$cost_{DMV_{dmq}} = \sum_{dmv_i \in DMV_{dmq}} ||dmv_i|| + k * ||\mathcal{R}'|| + ||\sigma_{\Sigma_{dmq}}\mathcal{R}||, \qquad (2)$$

where $\mathcal{R}'$ represents the portion of $\sigma_{\Sigma_{dmq}}\mathcal{R}$ which is not covered by any view from DMV_{dmq}, and k is the number of scans of $\mathcal{R}'$ required by a complete mining algorithm (e.g., *Apriori*) that has to be run to discover locally frequent itemsets in $\mathcal{R}'$.

In order to estimate the benefits from using multiple data mining views, let us compare the above cost with the cost of running a regular algorithm on the complete dataset. Assuming that the complete mining algorithm would need the same number k of dataset scans on the whole database and the portion of it not covered by materialized views, the materialized data mining views should be used if:

$$\sum_{dmv_i \in DMV_{dmq}} ||dmv_i|| + k * ||\mathcal{R}'|| < (k-1) * ||\sigma_{\Sigma_{dmq}}\mathcal{R}||. \qquad (3)$$

Since in practical applications we have $||dmv_i|| << ||\sigma_{\Sigma_i}\mathcal{R}||$, then the larger coverage of *dmq*'s source dataset by the views from DMV_{dmq}, the better performance of *dmq*'s execution. Therefore we look for such a DMV_{dmq} that provides the largest coverage of *dmq*'s source dataset.

Algorithm 2 (Materialized data mining view selection).
Input: A data sharing graph $DSG = (DMV \cup \{dmq\}, E)$, a set of distinct selection formulas S, a data mining query $dmq = (\mathcal{R}, a, \Sigma_{dmq}, \Phi_{dmq}, \beta_{dmq})$
Output: DMV_{dmq} providing the largest coverage of *dmq*'s source dataset.
Method:

```
      begin
 1.     $P \leftarrow \{s \in S | (dmq, s) \in E\}$
 2.     $SV \leftarrow \{dmv \in DMV | \forall s \in S, (dmv, s) \in E \Rightarrow s \in P\}$
 3.     $AV \leftarrow \{dmv \in SV | dmv = (\mathcal{R}, a, \Sigma, \Phi, \beta), \beta \leq \beta_{dmq}, \Phi \subseteq \Phi_{dmq}\}$
 4.     global $optCost \leftarrow +\infty$
 5.     global $OptViewSet \leftarrow \emptyset$
 6.     $scanViewSets(AV, \emptyset)$
 7.     return $OptViewSet$
      end

 8. procedure scanViewSets(V, Seed):
      begin
 9.     for all $dmv \in V$ do begin
10.       $newCost \leftarrow cost(Seed \cup \{dmv\}, dmq)$
11.       if $newCost < optCost$ then
12.         $optCost \leftarrow newCost$
13.         $optViewSet \leftarrow Seed \cup \{dmv\}$
          end if
14.       $V \leftarrow V - \{dmv\}$
15.       $OL \leftarrow \{dmv_j | \exists s, (dmv, s) \in E \wedge (dmv_j, s) \in E\}$
16.       $scanViewSets(V - OL, Seed \cup \{dmv\})$
      end
```

The above algorithm constructs a DMV_{dmq} providing the largest coverage of *dmq*'s source dataset. In step (1) we build the set P of distinct selection formulas for *dmq*. In step (2) we select materialized data mining views (SV) containing such data selection conditions that are relaxed with respect to *dmq*'s data selection conditions. In step (3) we select such views from SV that use *minsup* not above *dmq*'s *minsup* and use pattern selection conditions identical to or relaxed with respect to *dmq*. In step (4) we initialize a global variable to hold the best I/O cost found so far. In step (5) we initialize a global variable to hold the best set of materialized data mining views found so far. In step (6) we call the recursive procedure to generate all allowable subsets AV of the materialized data mining. In step (7) we return the optimal set of materialized data mining views for the execution of *dmq*. In step (8) the procedure called *scanViewSets* begins; it scans all allowable subsets of V, and it appends *Seed* to each of them. In step (9) a loop begins to iterate over all materialized data mining views from V. In step (10) we evaluate the I/O cost of executing *dmq* using the set of materialized data mining views. If the cost is lower that the best one found so far, then in steps (12)-(13) we update the global variables. In step (14) we eliminate the current view from the set not to process it again during recursive calls. In step (15) we select all materialized data mining views which share at least one distinct data selection formula with the current view. In step (16) we recursively call *scanViewSets* to append more views to the current set.

4 Experimental Results

In order to evaluate performance of frequent itemset discovery using a collection of materialized views, we performed several experiments on a Pentium II 433MHz PC with 128 MB of RAM. The experiments were conducted on the MSWeb[1] (Microsoft Anonymous Web Data) dataset from the UCI KDD Archive [5]. The dataset contained 32710 transactions (user sessions), the average size of transaction was 3 URLs, and the total number of different URLs in the dataset was 285. As a complete data mining algorithm we used our implementation of *Apriori*.

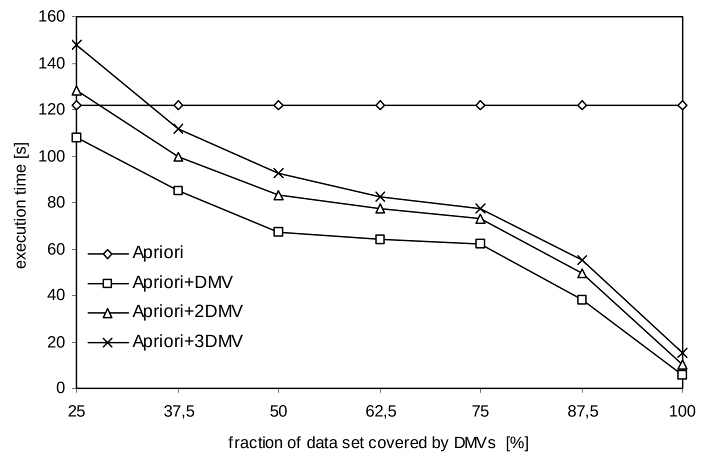

Fig. 3. Execution times for different numbers of used materialized views

Figure 3 presents the execution times for a different number of selected materialized views (1, 2, and 3) compared to the execution time of the complete

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

Apriori algorithm. For the cases involving materialized views we varied the fraction of the dataset covered by them from 25% to 100%. The minimum support threshold of the query to be answered was by 0.1 higher than the average minimum support threshold of the selected materialized views.

The experiments show that using materialized views reduces processing time if the views cover a significant fraction of the original query's dataset. Already for materialized views covering 37.5% of the source dataset, using up to 3 views paid off. The best performance was achieved when the selected views covered the whole dataset - in such cases there was no need to run *Apriori* at all.

Regarding the number of selected materialized views, we observe that for the same level of coverage, the smaller the number of views the better. One reason for performance degradation due to using a higher number of views is that using more views means higher I/O costs as more views have to be read from disk. This effect is particularly noticeable for small datasets like MSWeb. Another issue that might affect performance even in case of large datasets is the distribution of values in the dataset. If the distribution is strongly skewed, each of the views will introduce many locally frequent patterns, which then have to be counted (verified) in the whole dataset.

5 Conclusions

In this paper we discussed answering a frequent itemset query using a collection of materialized data mining views. The proposed method is independent of a particular mining algorithm, and is a generalization of the method from our previous work that was capable of using only one view.

We have presented: (1) the conditions on a set of materialized views that have to be fulfilled for the set to be useful, (2) the algorithm that uses results from a set of materialized views to answer a new frequent itemset query, (3) the algorithm that finds the subset of applicable materialized views that is optimal in terms of I/O costs.

The experiments confirm our theoretical analysis and show that using a set of materialized views is an efficient solution, provided that the views cover a significant part of the dataset.

References

1. Agrawal, R., Imielinski, T., Swami, A. (1993) Mining Association Rules Between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD Conference on Management of Data, Washington, D. C., 207–216
2. Agrawal, R., Srikant, R. (1994) Fast Algorithms for Mining Association Rules. Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 487–499

3. Baralis, E., Psaila, G. (1999) Incremental Refinement of Mining Queries. Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Florence, Italy, 173–182
4. Cheung, D. W.-L., Han, J., Ng, V., Wong, C. Y. (1996) Maintenance of discovered association rules in large databases: An incremental updating technique. Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA, 106–114
5. Hettich, S., Bay, S. D. (1999) The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science
6. Imielinski, T., Mannila, H. (1996) A Database Perspective on Knowledge Discovery. Communications of the ACM **39**(11), 58–64
7. Morzy, T., Wojciechowski, M., Zakrzewicz, M. (2000) Materialized Data Mining Views. Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000), Lyon, France, 65–74
8. Nag, B., Deshpande, P. M., DeWitt, D. J. (1999) Using a Knowledge Cache for Interactive Discovery of Association Rules. Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, San Diego, California, 244–253
9. Roussopoulos, N. (1998) Materialized Views and Data Warehouses. SIGMOD Record, **27**(1), 21–26
10. Zakrzewicz, M., Morzy, M., Wojciechowski, M. (2004) A Study on Answering a Data Mining Query Using a Materialized View. Proceedings of the 19th International Symposium on Computer and Information Sciences (ISCIS'04), Kemer - Antalya, Turkey, 493–502