Incremental Data Mining Using Concurrent Online Refresh of Materialized Data Mining Views

Mikołaj Morzy, Tadeusz Morzy, Marek Wojciechowski, and Maciej Zakrzewicz

Institute of Computing Science
Poznań University of Technology, Piotrowo 3A, 60-965
Poznań, Poland
{mmorzy,tmorzy,mwojciechowski,mzakrzewicz}@cs.put.poznan.pl

Abstract. Data mining is an iterative process. Users issue series of similar data mining queries, in each consecutive run slightly modifying either the definition of the mined dataset, or the parameters of the mining algorithm. This model of processing is most suitable for incremental mining algorithms that reuse the results of previous queries when answering a given query. Incremental mining algorithms require the results of previous queries to be available. One way to preserve those results is to use materialized data mining views. Materialized data mining views store the mined patterns and refresh them as the underlying data change.
Data mining and knowledge discovery often take place in a data warehouse environment. There can be many relatively small materialized data mining views defined over the data warehouse. Separate refresh of each materialized view can be expensive, if the refresh process has to re-discover patterns in the original database. In this paper we present a novel approach to materialized data mining view refresh process. We show that the concurrent on-line refresh of a set of materialized data mining views is more efficient than the sequential refresh of individual views. We present the framework for the integration of data warehouse refresh process with the maintenance of materialized data mining views. Finally, we prove the feasibility of our approach by conducting several experiments on synthetic data sets.

1 Introduction

Data mining, or knowledge discovery in databases, is a non-trivial process of finding valid, novel, useful, and ultimately understandable patterns and regularities in very large data volumes [5]. Data mining systems are quickly evolving from specific to general-purpose systems that are tightly coupled with the existing relational database technology. Integration is usually performed in the data warehouse, which serves high quality data to various mining techniques. Mining processing characteristics differs significantly from typical database workload. Hence, new methods of data mining query processing and optimization are being developed. One of these methods is incremental discovery of patterns.

The patterns discovered as the result of the execution of a data mining algorithm can be regarded as an answer to a sophisticated database query. A user defines the set of mined data using standard SQL commands and determines the parameters governing a given data mining algorithm. In response, relevant patterns are returned to the user for evaluation. Users usually do not achieve satisfying results immediately. It is an iterative process, where in each consecutive step the user evaluates the patterns and, suitably to the needs, expectations, and experience, modifies either the mined dataset, or algorithm parameters, or both. Because of this iterative and repetitive nature of mining processing, a data mining system must efficiently exploit the results of previous queries when fulfilling user requests. Data warehouses facing similar requirements in on-line analytical processing are materializing the results of queries as snapshots and rewrite incoming queries to use the materialized data. The same principle applies to data mining systems, where previously discovered patterns are stored in materialized data mining views and used to efficiently answer user queries.

The main problem in using materialized patterns gathered during mining is the freshness of the patterns. Each update of the source data can potentially invalidate some or all patterns stored in a materialized data mining view. This is particularly important in a data warehouse, where refresh of the source data happens regularly, often on a daily or even hourly basis. To cope with this problem, incremental mining techniques were proposed that aim at efficient maintenance of materialized patterns by running the base algorithm only on the difference set, and minimizing the number of full data reads necessary to validate the patterns. The main drawback of this approach is the fact, that the proposed methods refresh materialized data mining views separately, disregarding the properties of the data warehouse environment.

The refresh cycle of modern data warehouses becomes more and more frequent. As the result, the volume of the data loaded upon each refresh becomes relatively smaller as compared to the size of the entire data warehouse. On the other hand, the number of materialized data mining views defined in a data warehouse increases. In practice, materialized data mining views that span entire fact tables are rare. Instead of having a single large materialized data mining view, users define several well-focused views that cover a specific area of analysis (e.g., buyers of a particular product, purchases made during particular time of day, etc.). Usually, only a small subset of a huge fact table is used as the data source for such materialized data mining view. Consequently, the probability that a given tuple contained in the data loaded into a data warehouse upon refresh is likely to influence the patterns stored in a materialized data mining view is much smaller than assumed in traditional approaches. In such circumstances, using the entire load of new data and repeating incremental maintenance procedures for all materialized data mining views separately is a waste of resources.

We propose to tackle the problem of materialized data mining view maintenance from another perspective. Our solution is concurrent online refresh of a set of materialized data mining views. In our approach the maintenance of materialized data mining views becomes an integral part of the data warehouse refresh process. When a tuple is loaded into the warehouse, materialized data mining views that could become affected by that tuple are updated simultaneously. In this paper we present a framework for online maintenance of a set of materialized data mining views. We argue that our assumptions are reasonable with respect to practical data warehouse implementations. Our contribution is the following. We demonstrate the structure for fast lookup of candidate materialized data mining views that could become affected by an insert of a tuple. We present a novel algorithm for concurrent online incremental mining and we experimentally prove the feasibility of the proposed approach by comparing our algorithm with other algorithms. Experimental comparison of our algorithm with two algorithms proposed in the literature so far (Apriori and IUA) shows that our algorithm outperforms previous proposals.

The remainder of the paper is organized as follows. Following this paragraph we present the related work and definitions of basic terms used throughout the paper. The idea of using materialized results of previous data mining queries to answer subsequent user requests is presented in Section 2. In Section 3 we present a novel approach to the materialized pattern maintenance problem which consists in concurrent online refresh of materialized data mining views. Experimental evaluation of our approach is presented in Section 4. The conclusions are contained in Section 5.

1.1 Related Work

The problem of association rule mining was introduced in [2]. The paper identified the discovery of frequent itemsets as a key step in association rule mining. In [3] the authors presented basic algorithm called *Apriori*, which quickly became the seed of several other data mining algorithms. The first algorithm for maintaining discovered association rules using incremental technique, called FUP, was proposed in [4]. In [13] a new Incremental Updation Algorithm (IUA) was proposed. IUA minimized the number of full database scans to discover association rules in the updated data using the idea of the negative border of the collection of frequent itemsets [7].

The work on materialized views started in the 1980s. Multiple algorithms for view maintenance were developed [12]. Further research led to the creation of cost models for materialized view maintenance and applying views to enforce integrity constraints in databases. A summary of view maintenance techniques can be found in [6]. Materialized data mining views were first proposed in [10], and quickly became an important tool in data mining query optimization [8, 9, 14]. To the best of our knowledge the idea of concurrent online refresh of a set of materialized data mining views has not been presented yet.

1.2 Basic Definitions

Let $I = \{i_1, \ldots, i_n\}$ be a set of literals called items. Let D be a set of variable length transactions and $\forall T \in D : T \subseteq I$. We say that the transaction T supports an item x if $x \in T$. We say that the transaction T supports an itemset X

if it supports every element $x \in X$. The *support* of an itemset is the number of transactions supporting the itemset. The problem of discovering frequent itemsets consists in finding all itemsets with the support higher than user-defined minimum support threshold denoted as *minsup*. An itemset with the support higher than *minsup* is called a *frequent itemset*. Given a collection of frequent itemsets L. The negative border $NBd(L)$ of the collection L consists of all sets s_i, such that $s_i \notin L \wedge \forall s'_i \subset s_i : s'_i \in L$.

An association rule is an implication of the form $X \rightarrow Y$ where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. X is called the *body* of the rule whilst Y is called the *head* of the rule. Two measures represent statistical significance and strength of a rule. The *support* of a rule is the number of transactions that support $X \cup Y$. The *confidence* of a rule is the ratio of the number of transactions that support the rule to the number of transactions that support the head of the rule. The problem of discovering association rules consists in finding all rules with support and confidence higher than the user-specified thresholds of minimum support and confidence, called *minsup* and *minconf* respectively.

2 Data Mining Using Materialized Views

MineSQL [11] is a multi-purpose data mining query language which uses data mining queries to express data mining tasks. The syntax of MineSQL resembles standard SQL and provides excellent means of integration of data mining requests with the underlying database management system. MineSQL allows to issue commands that discover frequent itemsets, association rules, and sequential patterns. MineSQL uses additional data types (e.g. `SET`, `ITEMSET`, `RULE`) as well as operators and functions for those data types (e.g. `CONTAINS`, `BODY(x)`, `HEAD(x)`). The following data mining query discovers all association rules with support higher than 2.5% and confidence higher than 70%, which contain an item '*Bordeaux Pomerol*' in the body of the rule. Mining is performed over transactional data on premium customers for the 2nd half of the year 2004 .

```
MINE RULE r, HEAD(r), BODY(r)
FOR products FROM (
  SELECT SET(product) AS products
  FROM PurchaseFacts
  WHERE time_id >= '01.07.2004'
    AND time_id <= '31.12.2004'
    AND customer_type = 'Premium'
  GROUP BY transaction_id )
WHERE SUPPORT(r) > 0.025
  AND CONFIDENCE(r) > 0.7
  AND BODY(r) CONTAINS TO_SET('Bordeaux Pomerol');
```

In traditional databases a view defines a mapping function from a set of base relations to the derived relation. The function is computed upon each reference to the view. Views hide complex data structures from a user and provide an additional independence layer between an application and the underlying database schema. Changes occurring in the database schema are reflected only in the view definition with no impact on the end-user application. In order to avoid computational overhead, the contents of the view can be materialized in the database. The materialized copy of the data can be quickly accessed, thus bypassing expensive computation of the view. Data stored in a materialized view are not automatically refreshed when base relations change. Therefore, view maintenance techniques are necessary to reflect changes that occur in base relations of a materialized view. Often, modifications of base relations affect only a part of the materialized view. Incremental view maintenance techniques avoid recomputation of the entire view contents by determining the parts of the materialized view that should be updated.

Similarly to traditional database views, data mining views can be used to simplify application development, hide the complexity of data mining algorithms behind standard view interface, and enable incremental mining techniques by materializing results of data mining queries for further processing. Consider the following MineSQL statement that defines a materialized data mining view `v_saint_emilion`.

```
CREATE MATERIALIZED VIEW v_saint_emilion
REFRESH 7 AS
MINE RULE r, BODY(r), SUPPORT(r), CONFIDENCE(r)
FOR products FROM (
  SELECT SET(product) AS products
  FROM PurchaseFacts
  WHERE time_id >= '01.01.2004'
    AND time_id <= '31.12.2004'
  GROUP BY transaction_id
  HAVING AVG(price) >= 10 )
WHERE SUPPORT(r) > 0.025
  AND HEAD(r) CONTAINS TO_SET('Saint Emilion');
```

The definition of the view contains two classes of constraints: *database constraints* appear within the `WHERE` clause in the `SELECT` subquery, whereas *mining constraints* appear within the `WHERE` clause in the `MINE` statement. Database constraints delimit the part of the database that constitutes the source dataset. Mining constraints define patterns that are interesting to the user. Materialized data mining view not only separates the user from the technical details of the underlying mining algorithm, but provides the storage for discovered patterns. Every pattern in a materialized data mining view has a timestamp representing its creation time and validity period. One can provide the `REFRESH` clause that defines the period after which the contents of the materialized data mining view should be refreshed. Materialized views can be refreshed manually or automatically. The refresh of a materialized view could be performed by an incremental mining algorithm, or could involve the recomputation of the entire view.

The importance of materialized data mining views stems from the fact that the contents of the materialized data mining view can be used to efficiently answer a data mining query which is similar to the materialized view definition. Depending on the relations between a query and the view definition, several different mining methods are available. These methods include incremental mining, complementary mining, verifying mining, and full mining. Data mining query optimization using materialized data mining views is covered in [8, 14].

3 Concurrent Online Refresh of Materialized Data Mining Views

Data warehouse is an integrated collection of high-quality data supporting decision making. Based on this data, users can define multiple, possibly overlapping, collections of related data that serve as data sources for data mining queries. We argue that in typical applications users perform a focused selection of source data of interest and constraint their data mining activities to the selected subsets of the original data. We attribute this behavior to the fact that very large volumes of data produce patterns that are too general to be useful in analysis and decision making. Rather, users concentrate on smaller sets of data that are relevant to a given data mining query. After determining the subset of interesting data and setting the parameters of a mining algorithm, users can store their mining activity as a materialized data mining view and decide on the refresh frequency. Therefore, one should perceive a data warehouse as an environment for multiple different materialized data mining views that can be refreshed and maintained independently. Moreover, as changes to the data warehouse do not happen continuously over time, but are loaded in chunks during periodical data warehouse refresh, the refresh of materialized data mining views can be integrated with the process of the entire data warehouse refresh. It is worth noticing that during data warehouse refresh, when new tuples are loaded into base tables, they do not necessarily invalidate all materialized data mining views. Whether a tuple invalidates the patterns stored in a materialized data mining view or not, depends solely on the definition of the materialized data mining view, in particular, on the database constraints of the view.

In our implementation we are using a special index table to store the definitions of materialized data mining views. For all database tables that are used as the source for data mining views, table attributes are mapped to columns in the index table. Each materialized data mining view is described as a single row in the index table. In addition to columns representing attributes of relational tables, the index table stores the thresholds of minimum support and confidence provided in the view definition. If the definition of the materialized data mining view contains a database constraint defined on a base table attribute, this fact is reflected in the index table by inserting the relational operator ($=, \leq, \geq, \neq$,etc.) with the associated constant value into the appropriate column of the index table. A special symbol '*' is used if attributes of a base table are used in the materialized data mining view without any constraints. The index table is used to quickly decide, which of the materialized data mining views defined in the data warehouse are affected by the insertion of a given tuple. This is done by comparing the values in the inserted tuple with the constants stored in the appropriate attributes of the index table.

Each materialized data mining view is implemented as a relational table. The table contains both frequent itemsets constituting the answer to the data mining query, and the negative border of the collection of frequent itemsets. Each itemset is represented as a single row having two attributes: a numerical attribute containing the support of the itemset and a collection of items implemented as a nested table of varying length. This schema can be easily extended to support the storage of association rules. Given an association rule $X \rightarrow Y$. Both X and Y must be frequent itemsets. Therefore, both itemsets appear in the base relational table. The row representing the itemset X has an additional column `HEADS` which is a nested table of `head` objects. Each `head` object consists of the numerical confidence measure and the pointer to the itemset forming the head of the rule (in the above example the pointer points to the location of the itemset Y). Analogously, every itemset has an additional column `BODIES` implemented as a nested table of `body` objects. Again, each `body` object consists of the confidence measure of the rule and the pointer to the itemset forming the body of the rule. Pointers are either artificial primary keys or physical row addresses. Using bi-directional pointers to rule elements allows for fast lookup of rules containing a given itemset in the body or the head of the rule.

The algorithm for concurrent online refresh of materialized data mining views, denoted OUA for Online Updation Algorithm, proceeds as follows. The insertion of new tuples into the data warehouse base table triggers the verification procedure. First, definitions of all materialized data mining views defined on the updated base table are retrieved from the index table. Next, values of attributes of a newly inserted tuple are compared to the values of attributes used in database constraints of materialized data mining views. The comparison is performed using a special function which selects appropriate relational operators to test, whether the tuple satisfies all constraints defined in the definition of a materialized data mining view. If the comparison succeeds, the procedure updates the view.

Given a materialized data mining view $MDMV$ and a newly inserted tuple t that affects the view. Let L denote the collection of frequent itemsets present in the materialized data mining view $MDMV$. Let $NBd(L)$ denote the negative border of the collection of frequent itemsets L and let C_k denote the collection of candidate itemsets of the size k.

In the first step the set of items contained in the newly inserted tuple t is divided to form the collection of one-element candidate itemsets C_1. For all elements in C_1 the algorithm searches for itemsets $s \in L \cup NBd(L)$, such that $s = c$, and increases the support count of these itemsets. If, during this step, an itemset from the negative border $NBd(L)$ becomes frequent, it is moved to L and the negative border is expanded to reflect this move. Next, itemsets $C_1 \cap L$ are used to generate the set of candidate 2-itemsets C_2. Observe that only 2-itemsets contained in the inserted tuple are used to grow L and $NBd(L)$. This procedure repeats until no more candidate k-itemsets can be generated (candidates are generated using standard *apriori-gen* procedure of the *Apriori* algorithm). After processing all new tuples the algorithm checks if the negative border of the collection of frequent itemsets should be updated. This happens if there is a set that moved from the negative border to the collection of frequent itemsets. This step may require a full database scan.

Example 1. Given the materialized view $MDMV$ with $L = \{A, B, C, AB, AC\}$ and $NBd(L) = \{BC\}$. Let the newly inserted tuple $t = \langle A, B, C\rangle$. First, all elements of the tuple t are used to create the collection of candidate 1-itemsets $C_1 = \{A, B, C\}$. This collection is compared with $L \cup NBd(L)$ and the appropriate support counts are incremented. No itemsets are moved from the negative border to the set of frequent itemsets. Next, the set L_1 is determined as $L_1 = C_1 \cap L = \{A, B, C\}$. These itemsets are used to generate the set of candidate 2-itemsets $C_2 = \{AB, AC, BC\}$. Again, these itemsets are compared with $L \cup NBd(L)$ to increase appropriate support counts. As the result, the itemset BC is moved from $NBd(L)$ to L and the negative border $NBd(L)$ is expanded with the itemset ABC. As in previous step, the set L_2 is determined as $L_2 = C_2 \cap L = \{AB, AC, BC\}$. This procedure repeats until no new candidates can be generated. Support counts for itemsets from the expanded negative border are determined during additional database scan.

4 Experimental Results

All experiments were conducted on Dell Pentium M 1,4 GHz with 768 MB of RAM running Windows 2000 and Oracle 9*i*. Data sets were created using DBGen generator from the Quest Project [1]. Original database contained 100 000 transactions, the average size of the transaction was 40, and the number of different items was set to 100 000. For comparison we have chosen the basic Apriori algorithm (no incremental mining at all) and Incremental Updation Algorithm [13]. The size of the base table update varied from 500 to 5000 new transactions (i.e., from 0.5% to 5% of the original data volume). The percentage of the original base table covered by the materialized data mining view varied from 2.5% to 50%, the support threshold changed from 1.5% to 5%, the number of materialized data mining views that were simultaneously updated varied from 5 to 20.

The results of experiments are depicted in Figures 1-4. As expected, our algorithm works best when the number of materialized data mining views is large and the degree of coverage of base table is small. Again, we argue that this situation is typical for most applications using data mining techniques within the data warehouse environment. An important factor that affects the performance of our algorithm is the size of the update. For larger updates the cost of processing of each tuple separately surpasses the gain of not reading the update several times (especially when the number of concurrently updated materialized views is small). In such cases Incremental Updation Algorithm is a winner. We believe that this result is not discouraging, because we are observing a continuous shrinking of the data warehouse refresh window. Our algorithm is best suited

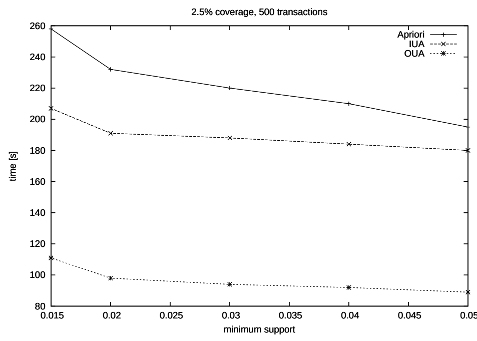

Fig. 1. time vs. support, 2.5% coverage

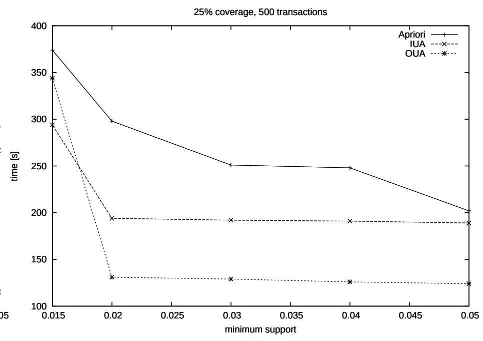

Fig. 2. time vs. support, 25% coverage

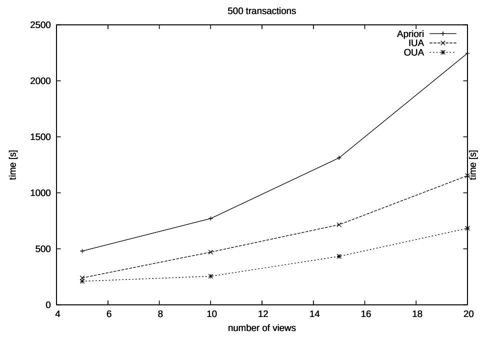

Fig. 3. time vs. number of views

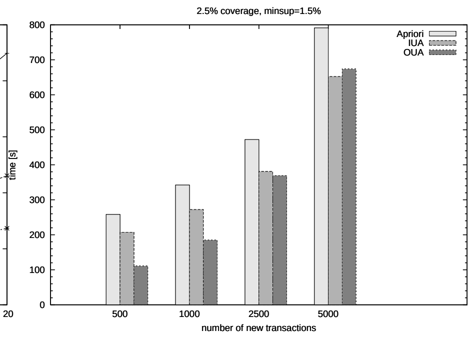

Fig. 4. time vs. update size

for frequently refreshed warehouses, where the contents of the data warehouse must be synchronized with operational databases on a daily or hourly basis.

5 Conclusions

In this paper we have presented Online Updating Algorithm, which implements a novel approach to materialized data mining view maintenance problem. Instead of performing separate refresh of a set of materialized data mining views we propose to update them simultaneously, during the data warehouse refresh process. Our algorithm outperforms previously proposed methods in environments where many materialized data mining views are defined over relatively small subsets of source data. We argue that this assumption holds in most practical applications, hence our algorithm provides improvement over other approaches.

References

1. R. Agrawal, M. J. Carey, C. Faloutsos, S. P. Ghosh, M. A. W. Houtsma, T. Imielinski, B. R. Iyer, A. Mahboob, H. Miranda, R. Srikant, and A. N. Swami. Quest:

A project on database mining. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, page 514, Minneapolis, Minnesota, may 1994. ACM Press.

2. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 207–216. ACM Press, 1993.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.
4. D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In S. Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 106–114. IEEE Computer Society, 1996.
5. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.
6. A. Gupta and I. S. Mummick. *Materialized Views: Techniques, Implementations, and Applications.* The MIT Press, 1999.
7. H. Mannila and H. Toivonen. An algorithm for finding all interesting sentences. In *Proc. of the 6th Int'l Conference on Database Theory*, pages 215–229, 1996.
8. M. Morzy, T. Morzy, and Z. Królikowski. Incremental association rule mining using materialized data mining views. In T. Yakhno, editor, *Advances in Information Systems, 3rd International Conference, ADVIS 2004, Izmir, Turkey, October 20-22, 2002, Proceedings*, volume 3261 of *Lecture Notes in Computer Sciences*, pages 77–87. Springer-Verlag, 2004.
9. M. Morzy, M. Wojciechowski, and M. Zakrzewicz. Cost-based sequential pattern query optimization in presence of materialized results of previous queries. In M. A. Klopotek, S. Wierzchon, and M. Michalewicz, editors, *Intelligent Information Systems 2002*, Advances in Soft Computing, pages 435–444. Physica-Verlag, jun 2002.
10. T. Morzy, M. Wojciechowski, and M. Zakrzewicz. Materialized data mining views. In D. A. Zighed, H. J. Komorowski, and J. M. Zytkow, editors, *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, volume 1910 of *Lecture Notes in Computer Science*, pages 65–74. Springer, 2000.
11. T. Morzy and M. Zakrzewicz. Sql-like language for database mining. In *1st East European Symposium on Advances in Databases and Information Systems, ADBIS 1997, St.Petersburg, Russia, September , 1997, Proceedings*, 1997.
12. N. Roussopoulos. Materialized views and data warehouses. *SIGMOD Record*, 27(1), 1998.
13. S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 263–266. AAAI Press, 1997.
14. M. Zakrzewicz, M. Morzy, and M. Wojciechowski. A study on answering a data mining query using a materialized view. In C. Aykanat, T. Dayar, and I. Korpeoglu, editors, *Computer and Information Sciences - ISCIS 2004, 19th International Symposium, Antalya, Turkey, October 27-29, 2003, Proceedings*, volume 3280 of *Lecture Notes in Computer Sciences*, pages 493–502. Springer-Verlag, 2004.