Optimizing a Sequence of Frequent Pattern Queries*

Mikolaj Morzy, Marek Wojciechowski, Maciej Zakrzewicz

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

Abstract. Discovery of frequent patterns is a very important data mining problem with numerous applications. Frequent pattern mining is often regarded as advanced querying where a user specifies the source dataset and pattern constraints using a given constraint model. A significant amount of research on efficient processing of frequent pattern queries has been done in recent years, focusing mainly on constraint handling and reusing results of previous queries. In this paper we tackle the problem of optimizing a sequence of frequent pattern queries, submitted to the system as a batch. Our solutions are based on previously proposed techniques of reusing results of previous queries, and exploit the fact that knowing a sequence of queries a priori gives the system a chance to schedule and/or adjust the queries so that they can use results of queries executed earlier. We begin with simple query scheduling and then consider other transformations of the original batch of queries.

1 Introduction

Discovery of frequent patterns is a very important data mining problem with numerous practical applications. The two most prominent classes of patterns are frequent itemsets [1] and sequential patterns [3]. Informally, frequent itemsets are subsets frequently occurring in a collection of sets of items, and sequential patterns are the most frequently occurring subsequences in sequences of sets of items.

Frequent pattern mining is often regarded as advanced querying where a user specifies the source dataset, the minimum frequency threshold (called *support*), and optionally pattern constraints within a given constraint model [7]. A significant amount of research on efficient processing of frequent pattern queries has been done in recent years, focusing mainly on constraint handling and reusing results of previous queries in the context of frequent itemsets and sequential patterns.

In this paper we tackle the problem of optimizing a sequence of frequent pattern queries, submitted to the system at the same time or within a short time window. Our approach is motivated by data mining environments working in batch mode, where users submit batches of queries to be scheduled for execution. However, the techniques discussed in the paper can also be applied to systems following the on-line

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

interactive mining paradigm if we allow the system to group the queries received within a given time window and process them as batches. Our solutions are based on previously proposed techniques of reusing results of previous frequent pattern queries, and exploit the fact that knowing a sequence of queries a priori gives the system a chance to schedule and/or adjust the queries so that they can use results of queries executed earlier. We begin with simple query scheduling and then discuss other possibilities of transforming the original batch of queries.

The paper is organized as follows. In Section 2 we review related work. Section 3 contains basic definitions regarding frequent pattern queries and relationships between them. In Section 4 we present our new technique of optimizing batches of frequent pattern queries. Section 5 contains conclusions and directions for future work.

2 Related Work

Multiple-query optimization has been extensively studied in the context of database systems (e.g., [13]). The idea was to identify common subexpressions and construct a global execution plan minimizing the overall processing time by executing the common subexpressions only once for the set of queries. Data mining queries could also benefit from this general strategy, however due to their different nature they require novel multiple-query processing methods.

Within the data mining context, multiple-query optimization has not drawn much attention so far. As an introduction to multiple data mining query optimization, we can regard techniques of reusing intermediate or final results of previous queries to answer a new query. Methods falling into that category are: incremental mining, caching intermediate query results, and reusing materialized results of previous queries provided that syntactic differences between the queries satisfy certain conditions.

Incremental mining was first studied in the context of frequent itemsets in [5], where the *FUP* algorithm was proposed. Incremental mining consist in efficiently discovering frequent patterns in an incremented dataset, exploiting previously discovered frequent patterns. After the pioneer *FUP* algorithm, several other incremental mining algorithms were proposed for itemsets and sequential patterns.

An interesting solution based upon the idea of reusing intermediate results of previous queries was proposed in [10] by introducing the concept of a knowledge cache that would keep recently discovered frequent itemsets along with their support value, in order to facilitate interactive and iterative mining. In [8], the authors postulated to cache only non-redundant itemsets like closed itemsets [11].

Syntactic differences between data mining queries, representing situations when one query can be efficiently answered using the results of another query, have been first analyzed in [4] for association rule queries. The authors identified three relationships between the queries: equivalence, inclusion, and dominance, and provided appropriate query processing algorithms exploiting the relationships.

In [9], we proposed to materialize results of frequent pattern queries rather than rule queries, motivated by the fact that generation of rules from patterns is a

straightforward and relatively inexpensive task. The results of previous queries were stored in the form of materialized data mining views. Syntactic differences between frequent pattern queries considered in the paper included one leading to the possibility of incremental mining, and one analogous to the inclusion relationship from [4]. Those syntactic differences and the corresponding relationships between pattern queries were later more thoroughly analyzed by us in [16] within an example constraint model for frequent itemset discovery.

To the best of our knowledge, the only two real multiple-query processing methods for frequent patterns are *Apriori Common Counting* (*ACC*) and *MineMerge*, proposed by us in [14] and [15] respectively. Unfortunately, both methods have significant drawbacks that limit their practical applications. *ACC* is bound to the *Apriori* algorithm [2], which is a serious limitation since *Apriori* has been outperformed by newer, pattern-growth algorithms (see [6] for an overview). Moreover, *ACC* requires more memory than its base algorithm – *Apriori*. On the other hand, *MineMerge* is not bound to a particular mining algorithm but has been proven non-deterministic, sometimes resulting in longer processing time than in case of sequential query processing. Therefore, our goal in this paper is introduction of a new method, not bound to any mining methodology, having memory requirements not greater that applied base mining algorithm, and guaranteeing performance gains, at least under certain assumptions.

3 Frequent Pattern Queries

In this section, we provide a universal definition of a frequent pattern query, and review the possibilities of reusing frequent pattern queries' results. We generalize the definitions and methods introduced by us for frequent itemset mining [16], and reformulate them so that they can serve as a basis for modeling batches of frequent pattern queries and development of batch optimization (scheduling) techniques.

3.1 Basic Definitions

Definition 1 (Frequent pattern query and its predicates). A *frequent pattern query* is a tuple $fpq = (R, a, \Sigma, \Phi, \beta)$, where R is a relation, a is an attribute of R, Σ is a condition involving the attributes of R (called *database predicate*), Φ is a condition involving discovered patterns (called *pattern predicate*), β is the minimum support threshold. The result of the *fpq* is a set of patterns discovered in $\pi_a \sigma_\Sigma R$, satisfying Φ, and having support $\geq \beta$.

Example 1. Given the database relation $R_1(a_1, a_2)$, where a_2 is a set-valued attribute and a_1 is of integer type. The frequent pattern query $fpq_1 = (R_1, "a_2", "a_1>5", "|itemset|<4", 3\%)$ describes the problem of discovering frequent itemsets in the set-valued attribute a_2 of the relation R_1. The frequent itemsets with support of at least 3% and length less than 4 are discovered in the collection of records having $a_1>5$.

3.2 Relationships Between Frequent Pattern Queries

Definition 2 (Identical queries). Two frequent pattern queries fpq_1 and fpq_2 are *identical* if they both operate on the same attribute a of the same relation R and $\Sigma_1 = \Sigma_2$ and $\Phi_1 = \Phi_2$ and $\beta_1 = \beta_2$.

We assume the existence of some canonical form for database predicates and pattern predicates, to which all query predicates will be transformed before any optimization takes place. Thus, we can assume that two queries will guarantee to return the same results for any database instance only if the queries are identical.

Definition 3 (Data set inclusion). Given two frequent pattern queries $fpq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $fpq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. We say that fpq_1 *includes data set* of fpq_2 (denoted as $fpq_2 \subseteq_d fpq_1$) if for each possible instance of the relation R, $\sigma_{\Sigma 2}R \subseteq \sigma_{\Sigma 1}R$.

Definition 4 (Pattern set inclusion). Given two frequent pattern queries $fpq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $fpq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. We say that fpq_1 *includes pattern set* of fpq_2 (denoted as $fpq_2 \subseteq_p fpq_1$) if for each possible instance of the relation R, all the patterns returned by fpq_2 will also be returned by fpq_1, and for each pattern returned by both queries, its counted support value will be the same for both queries.

In terms of query predicates, fpq_1 includes pattern set of fpq_2 if $\beta_1 \leq \beta_2$ and Φ_1 is a relaxation of Φ_2 (denoted as $\Phi_2 \subseteq \Phi_1$). (Formal definition of such relaxation will depend on a particular pattern constraint model. The only requirement is that the relaxation relationship should be a partial order on pattern sets. See [16] for an example constraint model.)

It should be noted that the above relationships between frequent pattern queries are defined in terms of query predicates, independently of the contents of the mined database. Therefore, we can assume that for a given particular constraint model and a given frequent pattern query language, the system will be able to discover the relationships between the queries within a batch, just by analyzing the syntactic differences between the queries. Although, for a flexible constraint model and/or query language, the task might not be trivial, it is considered an implementation issue and as such is beyond the scope of this paper.

Example 2. Given the database relation $R_1(a_1, a_2)$ from Example 1 and two frequent pattern queries: fpq_1 = (R_1, *"a_2"*, *"a_1>5"*, *"|itemset|<4"*, *4%*) and fpq_2 = (R_1, *"a_2"*, *"a_1<3"*, *"|itemset|<5"*, *2%*), we have $fpq_1 \subseteq_p fpq_2$, and no data set inclusion relationship between the queries.

3.3 Reusing Results of Previous Frequent Pattern Queries

According to the analysis from [16] a frequent pattern query fpq_2 can be efficiently answered using known (materialized) results of another query fpq_1 provided that $fpq_1 \subseteq_d fpq_2$ (i.e., fpq_2 operates on an incremented data set) and $fpq_2 \subseteq_p fpq_1$ (i.e., the pattern selection condition and the minimum support threshold of fpq_2 are not more restrictive than those of fpq_1).

The general algorithm for answering fpq_2 using the results of fpq_1, where $fpq_1 \subseteq_d fpq_2$ and $fpq_2 \subseteq_p fpq_1$, consists of two steps:

1) Result Filtering (RF) by removing the patterns returned by fpq_1 that do not satisfy the pattern selection condition and the minimum support threshold of fpq_2.
2) Incremental Mining (IM) using pattern selection condition and the minimum support threshold of fpq_2, treating the data set of fpq_2 as incremented data set of fpq_1 for which the patterns of fpq_2's interest are known from the previous step.

It should be noted that there are three particular cases, in which one of or even both the above steps can be omitted:

- If $\beta_1=\beta_2$ and $\Phi_1=\Phi_2$ then the filtering step is not needed;
- If $\Sigma_1=\Sigma_2$ then the incremental mining step is not needed;
- If $\Sigma_1=\Sigma_2$ and $\beta_1=\beta_2$ and $\Phi_1=\Phi_2$ then the results of fpq_2 are equal to the results of fpq_1 and therefore neither filtering nor incremental mining is needed.

Regarding the implementation details and costs of the two steps of the above query result reusing algorithm, we observe that the first step (RF) is a simple scan of the query results, inexpensive both in terms of computations (simple conditions on patterns) and I/O (query results are typically several orders of magnitude smaller than the queries' source dataset). As for the second step (IM), it is obvious that different incremental mining techniques can be (or have already been) developed for various types of patterns, constraint models, and mining methodologies. However, as a reference incremental pattern mining method, we can regard the partition-based incremental mining technique described in [16], exploiting the well-known ideas of partition-based mining [12].

The partition-based incremental frequent pattern mining technique logically divides the database into two partitions: (1) the records covered by the query fpq_1 ($\sigma_{\Sigma 1}R$) – for this partition the locally frequent patterns are known, (2) the records covered by the query fpq_2, and not covered by fpq_1 ($\sigma_{\Sigma 2}R$ - $\sigma_{\Sigma 1}R$). The method begins with discovering patterns locally frequent in the second partition. Next, based on the property of partition-based mining, locally frequent patterns from both partitions are used as candidate patterns for the whole fpq_2's data set ($\sigma_{\Sigma 2}R$), and counted in one scan of the data set.

In typical scenarios, incremental mining is more efficient that running a complete mining algorithm, and result filtering is significantly more efficient than incremental mining. Therefore, if for a given query there are results of a previous query that can be reused, the system should reuse them rather than run a complete mining algorithm. If more than one query's results are applicable, the system should first look for the possibility of Result Filtering (on the smallest available pattern set), and then, if RF is not possible, the system should opt for Incremental Mining or the combination of RF and IM involving the smallest increment of the dataset.

4 Optimizing Batches of Frequent Pattern Queries

The problem of optimizing batches of frequent pattern queries can be informally defined as follows: Given a batch (a sequence) of frequent pattern queries, find the execution plan that minimizes the total execution time of the whole batch. In this paper, we consider optimization techniques based on the idea of reusing some query's results to answer other queries. Within this framework, we develop a novel multiple-query optimization method for batches of frequent pattern queries starting with simple query scheduling, and then considering other transformations of the original batch. We assume that the batches of queries to be optimized contain no duplicates. Elimination of duplicates should be one of the pre-processing steps, right after the transformation of queries' predicates to the canonical form used by the system, which is required to determine the relationships between the queries.

4.1 Query Scheduling

For the purpose of modeling batches of frequent pattern queries, let us start with a formal definition of the relationship between queries capturing the possibility of reusing other queries' results:

Definition 5 (Result reusing). Given two frequent pattern queries: $fpq_1=(R, a, \Sigma_1, \Phi_1, \beta_1)$ and $fpq_2=(R, a, \Sigma_2, \Phi_2, \beta_2)$. We say that fpq_2 *can reuse results* of fpq_1 (denoted as $fpq_1 \rightarrow fpq_2$) if $\sigma_{\Sigma 1}R \subseteq \sigma_{\Sigma 2}R$ and $\Phi_2 \subseteq \Phi_1$ and $\beta_1 \leq \beta_2$.

Theorem 1. The relationship of result reusing is a partial order on a set (batch) of frequent pattern queries.

Proof. The relationship of result reusing is defined upon three partial order relationships on query predicates. To prove that the relationship of result reusing is also a partial order, we have to prove that it is reflexive, anti-symmetric, and transitive (from the definition of partial order). The three properties of the relationship of result reusing can be derived from its definition and inherent properties of the partial orders upon which it is built in the following way:

- $\sigma_{\Sigma 1}R \subseteq \sigma_{\Sigma 1}R \wedge \Phi_1 \subseteq \Phi_1 \wedge \beta_1 \leq \beta_1 \Rightarrow fpq_1 \rightarrow fpq_1$ (proof of reflexivity);
- $(fpq_1 \rightarrow fpq_2 \wedge fpq_2 \rightarrow fpq_1) \Rightarrow (\sigma_{\Sigma 1}R \subseteq \sigma_{\Sigma 2}R \wedge \Phi_2 \subseteq \Phi_1 \wedge \beta_1 \leq \beta_2 \wedge \sigma_{\Sigma 2}R \subseteq \sigma_{\Sigma 1}R \wedge \Phi_1 \subseteq \Phi_2 \wedge \beta_2 \leq \beta_1) \Rightarrow (\sigma_{\Sigma 1}R = \sigma_{\Sigma 2}R \wedge \Phi_2 = \Phi_1 \wedge \beta_1 = \beta_2) \Rightarrow (\Sigma_1 = \Sigma_2 \wedge \Phi_2 = \Phi_1 \wedge \beta_1 = \beta_2) \Rightarrow (fpq_1 = fpq_3)$ (proof of anti-symmetry);
- $(fpq_1 \rightarrow fpq_2 \wedge fpq_2 \rightarrow fpq_3) \Rightarrow (\sigma_{\Sigma 1}R \subseteq \sigma_{\Sigma 2}R \wedge \Phi_2 \subseteq \Phi_1 \wedge \beta_1 \leq \beta_2 \wedge \sigma_{\Sigma 2}R \subseteq \sigma_{\Sigma 3}R \wedge \Phi_3 \subseteq \Phi_2 \wedge \beta_2 \leq \beta_3) \Rightarrow (\sigma_{\Sigma 1}R \subseteq \sigma_{\Sigma 3}R \wedge \Phi_3 \subseteq \Phi_1 \wedge \beta_1 \leq \beta_3) \Rightarrow (fpq_1 \rightarrow fpq_3)$ (proof of transitivity).

Based on the above result reusing relationship, we propose the initial multiple-query optimization method for pattern queries, consisting in scheduling the batch of queries according to the result reusing relationship.

Algorithm 1 (Multiple-Query Optimization Using Query Scheduling)
Input: a set of pattern queries $FPQ = \{ fpq_1, fpq_2 , \ldots, fpq_n\}$ searching for frequent patterns in the *a* attribute of the database relation *R*
Output: results of queries from *FPQ*

1. sort *FPQ* according to the result reusing relationship to form a schedule $SFPQ = (sfpq_1, sfpq_2 , \ldots, sfpq_n)$ where for each $fpq_i \in FPQ$ there exist $sfpq_j \in SFPQ$ such that $fpq_i = sfpq_j$ and for each pair $sfpq_i$, $sfpq_j$ of queries in *SFPQ*: $sfpq_i \rightarrow sfpq_j \Rightarrow i < j$;
2. **for** i := 1 **to** n **do**
3. $MPQ = \{ sfpq_k : sfpq_k \rightarrow sfpq_i\}$;
4. **if** $MPQ = \varnothing$ **then**
5. execute $sfpq_i$ using a complete mining algorithm;
6. **else**
7. select $mpq \in MPQ$ for which the estimated cost of reusing its results to answer $sfpq_i$ is minimal; /* see Section 3.3 */
8. execute $sfpq_i$ reusing the results of *mpq*; /* RF + IM, RF, or IM */
9. **end if**;
10. **end for**;

Rationale: Sorting the queries according to the result reusing relationship guarantees that for each query $sfpq_i$ all the queries whose results $sfpq_i$ can reuse will be executed earlier. Thus, the algorithm maximizes the chances of efficiently answering the queries using available results of previous queries. (Note that since the result reusing relationship is a partial order, a topological sort algorithm has to be used, and in general more than one optimal schedule is possible.)

4.2 Query Scheduling with Addition of Intermediate Queries

Algorithm 1 can be regarded as an initial solution that optimizes processing of the batch of queries by introducing a query scheduling step. To identify further optimization possibilities, let us model a batch of pattern queries as a directed graph, in which the nodes represent queries and the edges represent the possibility on reusing the results of one query by another query.

Definition 6 (Query Reusing Graph). A directed graph $QRG = (V,E)$ is a *query reusing graph* for the set of frequent pattern queries *FPQ* if and only if $V = FPQ$, $E = \{(fpq_i, fpq_j) \mid fpq_i, fpq_j \in FPQ \wedge fpq_i \rightarrow fpq_j \wedge (!\exists\ fpq_k \in FPQ$ such that $fpq_i \rightarrow fpq_k \wedge fpq_k \rightarrow fpq_j)\}$.

Let us consider a database relation $R_1(a, b)$ and an example batch of frequent pattern queries $FPQ_1 = \{fpq_1, fpq_2, fpq_3, fpq_4, fpq_5, fpq_6\}$, where fpq_1=(R_1, *a*, *"10<b<20"*, *"true"*, *1%*), fpq_2=(R_1, *a*, *"10<b<30"*, *"length(pattern)<3"*, *2%*), fpq_3=(R_1, *a*, *"10<b<30"*, *"true"*, *5%*), fpq_4=(R_1, *a*, *"10<b<30"*, *"length(pattern)<4"*, *4%*), fpq_5=(R_1, *a*, *"10<b<30"*, *"true"*, *3%*), fpq_6=(R_1, *a*, *"0<b<20"*, *"true"*, *1%*). Figure 1 shows the query reusing graph for the batch of frequent pattern queries FPQ_1. To

support the analysis of possible optimizations, edges of the graph have been labeled with corresponding query reusing methods.

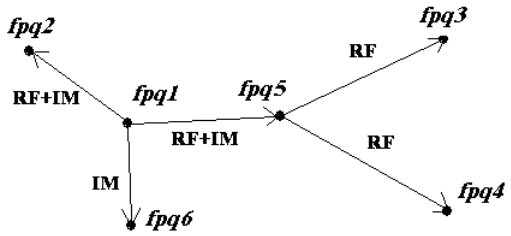

Fig. 1. Sample query reusing graph

Let us look at the queries fpq_2 and fpq_5 which can be answered using the results of fpq_1 in two steps: RF (using a different support threshold for each of the two queries) and IM (with exactly the same increment of the data set for the two queries). For a single query, if both RF and IM are required, it is more beneficial to start with RF and then run IM with the more restrictive pattern predicate and support threshold. However, if we know that more than one query will require the IM task on the same incremented data set as one of its execution steps, then typically it should be better to start with the IM step using the pattern predicate and support threshold that will allow all the involved queries to reuse the results of that IM step using RF procedures.

Identified common IM tasks can be represented as appropriate intermediate queries added to the original batch. Obviously, in this case the system will have to answer more queries than requested by users but as long as the total number of IM steps for the batch is reduced, the overall execution time of the batch should be shortened. (Recall that RF is typically by several orders of magnitude more efficient than IM.)

For the example batch FPQ_1 whose query reusing graph is presented in Fig. 1, we can provide the opportunity for reducing the number of executed IM tasks by adding an extra query fpq_7=(R_1, *a*, *"10<b<30"*, *"true"*, *2%*). Figure 2 presents the query reusing graph for the extended batch $FPQ_1' = FPQ_1 \cup \{fpq_7\}$.

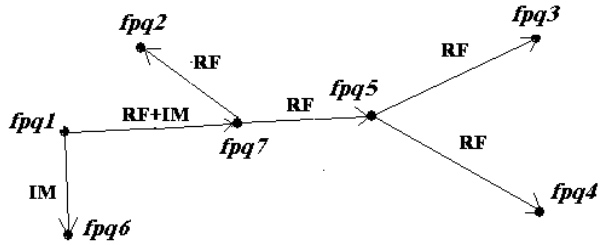

Fig. 2. Query reusing graph after the addition of the intermediate query

In general, such an intermediate query should have the same database constraint as the queries whose processing it going to improve, the support threshold equal to the

minimal support threshold among the queries, and pattern predicate being a logical alternative of the queries' pattern predicates. Based on the above observation, below we present an improved batch processing algorithm as an extension of Algorithm 1:

Algorithm 2 (Multiple-Query Optimization Using Query Scheduling with Intermediate Queries)
Input: a set of pattern queries $FPQ = \{ fpq_1, fpq_2, \ldots, fpq_n\}$ searching for frequent patterns in the *a* attribute of the database relation *R*
Output: results of queries from *FPQ*

1. **for each** $fpq_i \in FPQ$ **do**
2. $IMQ_i = \{ fpq_k : fpq_i \rightarrow fpq_k \wedge \Sigma_k \neq \Sigma_i \wedge$ for all $fpq_x, fpq_y \in IMQ_i : \Sigma_x = \Sigma_y\}$;
3. **if** $|IMQ_i| > 1$ **then**
4. $FPQ := FPQ \cup \{(R, a, \Sigma_{IMQ}, \Phi_{IMQ}, \beta_{IMQ})\}$, where Σ_{IMQ} is the database predicate of queries from IMQ_i, Φ_{IMQ} is the logical alternative of pattern predicates of all queries from IMQ_i, β_{IMQ} is the minimal support threshold among the queries from IMQ_i;
5. **end if**;
6. **end for**;
7. execute Algorithm 1 for *FPQ*

Rationale: An appropriate intermediate query is added for each set of queries that can reuse results of the same query using IM, provided that the set contains more than one query. As explained earlier, addition of each intermediate query to the batch reduces the number of IM tasks in the execution plan generated for the batch, which are typically much more costly than RF tasks.

4.3 Memory Management for Batch Execution

According to Algorithms 1 and 2, each of the pattern queries from a batch is executed using one of the three following methods: RF, IM, or complete mining. Taking into account that: (1) the most memory-consuming step of IM is execution of a base complete mining algorithm on the increment of the data set, and (2) RF can filter the patterns reading them from the disk one by one, we can say that memory requirements of our batch processing algorithms are not greater than in case of using a complete mining algorithm for all the queries in a batch, which is a desirable property.

Nevertheless, if possible within the memory limits, it will be beneficial for our technique to keep in main memory the results of queries than can be reused by some of the next queries (according to the generated schedule). As frequent pattern query results are typically much smaller than main memory structures used by pattern mining algorithms, such result caching introduces a negligible memory overhead. Moreover, once the system determines that the results of any of the previously executed queries cannot be reused by any queries to be executed later, the query's results can be removed from main memory, thus reducing the memory consumption.

5 Conclusions

In this paper we considered the problem of optimizing batches of frequent pattern queries. We presented a novel optimization technique based on techniques of reusing results of previous queries, previously proposed in literature. Our method exploits the fact that knowing a sequence of queries a priori gives the system a chance to schedule and/or adjust the batch of queries maximizing for each query the possibilities of reusing results of queries executed earlier.

The method proposed in this paper was motivated by data mining systems working in batch mode. In the future, we plan to focus on multiple-query optimization techniques oriented towards interactive systems, allowing dynamic addition of new queries to the set of currently optimized pattern queries.

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., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
3. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. 11th ICDE Conf. (1995)
4. Baralis E., Psaila G.: Incremental Refinement of Mining Queries. Proceedings of the 1st DaWaK Conference (1999)
5. Cheung D.W., Han J., Ng V., Wong C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proc. of the 12th ICDE (1996)
6. Han J., Pei J.: Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. SIGKDD Explorations, December 2000 (2000)
7. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
8. Jeudy B., Boulicaut J-F.: Using condensed representations for interactive association rule mining. Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (2002)
9. Morzy T., Wojciechowski M., Zakrzewicz M.: Materialized Data Mining Views. Proceedings of the 4th PKDD Conference (2000)
10. Nag B., Deshpande P.M., DeWitt D.J.: Using a Knowledge Cache for Interactive Discovery of Association Rules. Proc. of the 5th KDD Conference (1999)
11. Pasquier N., Bastide Y., Taouil R., Lakhal L.: Discovering frequent closed itemsets for association rules. Proc. 7th Int'l Conf. On Database Theory (1999)
12. Savasere A., Omiecinski E., Navathe S.: An Efficient Algorithm for Mining Association Rules in Large Databases, Proc. 21th Int'l Conf. Very Large Data Bases (1995)
13. Sellis T.: Multiple-query optimization. ACM Transactions on Database Systems, Vol. 13, No. 1 (1988)
14. Wojciechowski M., Zakrzewicz M.: Evaluation of Common Counting Method for Concurrent Data Mining Queries. Proc. of the 7th ADBIS Conference (2003)
15. Wojciechowski M., Zakrzewicz M.: Evaluation of the Mine Merge Method for Data Mining Query Processing. Proc. of the 8th ADBIS Conference (2004)
16. Zakrzewicz M., Morzy M., Wojciechowski M.: A Study on Answering a Data Mining Query Using a Materialized View. Proceedings of the 19th ISCIS Conference (2004)