Cost-based Sequential Pattern Query Optimization in Presence of Materialized Results of Previous 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. Data mining is very often regarded as an interactive and iterative process. Users interacting with the data mining system specify the class of patterns of their interest by means of data mining queries involving various types of constraints. It is very likely that a user will execute a series of similar queries, before he or she gets satisfying results. Unfortunately, data mining algorithms currently available suffer from long processing times, which is unacceptable in case of interactive mining. One possible solution, applicable in certain cases, is exploiting materialized results of previous queries when answering a new query. In this paper we discuss cost-based data mining query optimization in presence of materialized results of previous queries, focusing on one of the popular data mining techniques, called discovery of sequential patterns.

Keywords: data mining, sequential patterns, query optimization

1 Introduction

Data mining aims at discovery of useful patterns from large databases or warehouses. One of the well-known data mining methods is sequential pattern discovery introduced in [2]. Informally, sequential patterns are the most frequently occurring subsequences in sequences of sets of items. Typical sequential pattern mining algorithms discover all patterns whose support exceeds a user-specified threshold. Some of them allow users to specify time constraints [10] to be used when checking if a given source sequence contains a given pattern.

From a user’s point of view, data mining can be seen as an interactive and iterative process of advanced querying: a user specifies the source dataset and the requested class of patterns, the system chooses the appropriate data mining algorithm and returns discovered patterns to the user [5][6]. A user interacting with a data mining system has to specify several constraints on patterns to be discovered. However, usually it is not trivial to find a set of constraints leading to the satisfying set of patterns. Thus, users are likely to execute a series of similar data mining queries before they find what they need. Unfortunately, data mining algorithms require long processing times, which makes such interaction difficult.

One possible solution to that problem is exploiting materialized results of previous queries when answering a new query [3][7][9]. A data mining system should be able to determine which materialized query results can be used to answer the current query, and then to choose the one leading to the shortest response time. In has been observed [3] that the three particularly interesting relationships between two mining queries DMQ_1 and DMQ_2 extracting patterns from the same data are equivalence, inclusion, and dominance. The three relationships refer to results of the queries, not to their syntax, and are interesting since they represent situations, where one data mining query can be efficiently answered using the results of another query with no actual mining process. Equivalence, inclusion, and dominance relationships were introduced in the context of association rules. Nevertheless, they are general relationships applicable to many pattern types and constraint models.

Previous research on exploiting materialized patterns focused on identification of queries whose materialized results can be used to answer the current query. It has been shown experimentally that using materialized results of one of the previous queries is usually much more efficient than running a complete mining algorithm. However, none of the works addressed the problem of estimating the cost of answering a data mining query using materialized results of another query. Cost estimation is necessary in order to choose the optimal query answering plan when many possible strategies are applicable. In this paper, we discuss cost-based sequential pattern query optimization in the presence of materialized results of previous sequential pattern queries. The only goal of the optimization that we consider is minimizing the query execution time. We build on our previous work [11] where we identified situations in which one sequential pattern query can be answered using the results of another sequential pattern query. In this paper, we discuss strategies that can be used by a data mining query optimizer exploiting materialized results of previous queries. We provide cost functions for query answering algorithms exploiting materialized patterns. These cost functions are then used to choose an optimal (in terms of the execution time) query execution plan when many applicable materialized sets of patterns are available.

1.1 Related Work

To facilitate interactive and iterative pattern discovery, [9] proposed to materialize patterns discovered with the least restrictive selection criteria, and answer incoming queries by filtering the materialized pattern collection. In [7], the idea of caching intermediate results of association rule queries was discussed. In the approach, materialization of frequent itemsets instead of rules was proposed. However, in some cases it was required to materialize also some of the infrequent itemsets.

Cost-based query optimization is widely used in database management systems (see e.g. [4] for a review). One of the techniques used by query optimizers is exploiting results of previous queries available in the form of materialized views (see e.g. [8]). The cost-based optimizer chooses the query execution plan with the lowest estimated cost. The cost of a given execution strategy is estimated using known cost functions for the algorithms being used and certain statistics maintained for the database.

2 Basic Definitions

2.1 Sequential Patterns

Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $<X_1\ X_2\ ...\ X_n>$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. The *size* of a sequence is the number of items in the sequence. The *length* of a sequence is the number of elements in the sequence. Let D be a set of variable length sequences (called *data-sequences*), where for each sequence $S = <X_1\ X_2\ ...\ X_n>$, a timestamp is associated with each X_i.

With no time constraints we say that a sequence $X = <X_1\ X_2\ ...\ X_n>$ is *contained* in a data-sequence $Y = <Y_1\ Y_2\ ...\ Y_m>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We call $<Y_{i1}\ Y_{i2}\ ...\ Y_{in}>$ an *occurrence* of X in Y. We consider the following user-specified time constraints while looking for occurrences of a given sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), and time window that allows a group of consecutive elements of a data-sequence to be merged and treated as a single element as long as their timestamps are within the user-specified *window-size*.

The support of a sequence $<X_1\ X_2\ ...\ X_n>$ in D is the fraction of data-sequences in D that contain the sequence. A *sequential pattern* is a sequence whose support in D is above the user-specified threshold.

2.2 Relationships between Results of Data Mining Queries

Two data mining queries are *equivalent* if for all datasets they both return the same set of patterns and the values of statistical significance measures (e.g. support) for each pattern are the same in both cases. A data mining query DMQ_1 *includes* a data mining query DMQ_2 if for all datasets each pattern in the results of DMQ_2 is also returned by DMQ_1 with the same values of the statistical significance measures. A data mining query DMQ_1 *dominates* a data mining query DMQ_2 if for all datasets each pattern in the results of DMQ_2 is also returned by DMQ_1, and for each pattern returned by both queries its values of the statistical significance measures evaluated by DMQ_1 are not less than is case of DMQ_2.

Equivalence is a particular case of inclusion, and inclusion is a particular case of dominance. Equivalence, inclusion, and dominance meet the transitivity property.

If for a given query, results of a query equivalent to it, including it, or dominating it are available, the query can be answered without running a costly mining algorithm. In case of equivalence no processing is required, since the queries have the same results. In case of inclusion, one scan of the materialized query results is necessary to filter out patterns that do not satisfy constraints of the included query. In case of dominance, one scan of the source dataset is necessary to evaluate the statistical significance of materialized patterns (filtering out the patterns that do not satisfy constraints of the dominated query is also required).

3 Sequential Pattern Queries

In constraint-based sequential pattern mining, we identify three classes of constraints: database, pattern, and time constraints. Database constraints are used to specify the source dataset. Pattern constraints specify which patterns are interesting and should be returned by the query. Finally, time constraints influence the process of checking whether a given data-sequence contains a given pattern.

The basic formulation of the sequential pattern discovery problem introduces three time constraints: max-gap, min-gap, and time window, and assumes only one pattern constraint (expressed by means of the minimum support threshold). We model pattern constraints as complex Boolean predicates having the form of a conjunction of basic Boolean predicates of types presented below:

- π(**SPG**, α, pattern) – true if pattern support is greater than α, false otherwise;
- π(**SL**, α, pattern) – true if pattern size is less than α, false otherwise;
- π(**SG**, α, pattern) – true if pattern size is greater than α, false otherwise;
- π(**LL**, α, pattern) – true if pattern length is less than α, false otherwise;
- π(**LG**, α, pattern) – true if pattern length is greater than α, false otherwise;
- π(**C**, β, pattern) – true if β is a subsequence of the pattern, false otherwise;
- π(**NC**, β, pattern) – true if β is not a subsequence of the pattern, false otherwise.

Analyzing syntactic differences between sequential pattern queries leading to equivalence, inclusion, and dominance relationships between the queries, in [11] we identified two useful relationships regarding pattern and time constraints of sequential pattern queries, which can be informally defined as follows:

– *DMQ_2 extends pattern constraints* of DMQ_1 if pattern constraints of DMQ_1 can be transformed into pattern constraints of DMQ_2 by appending new basic Boolean pattern predicates or replacing basic Boolean pattern predicates with more selective predicates of the same types.

– *DMQ_2 extends time constraints* of DMQ_1 if it restricts at least one of the time parameters and does not relax any time parameters.

The following three theorems (proved in [11]) capture the influence of syntactic differences between queries on the differences between query results:

Theorem 1 Let *DMQ_1* and *DMQ_2* be two sequential pattern queries, operating on the same dataset and having the same time constraints. If *DMQ_2* extends pattern constraints of *DMQ_1*, then *DMQ_1* includes *DMQ_2*.
Theorem 2 Let *DMQ_1* and *DMQ_2* be two sequential pattern queries, operating on the same dataset and having the same pattern constraints. If *DMQ_2* extends time constraints of *DMQ_1*, then *DMQ_1* dominates *DMQ_2*.
Theorem 3 Let *DMQ_1* and *DMQ_2* be two sequential pattern queries, operating on the same dataset. If *DMQ_2* extends pattern constraints of *DMQ_1* and *DMQ_2* extends time constraints of *DMQ_1*, then *DMQ_1* dominates *DMQ_2*.

4 Cost Analysis of Sequential Pattern Query Execution Plans Exploiting Materialized Results of Another Query

Given a sequential pattern query *DMQ* and materialized results of a sequential pattern query *DMQ_V* operating on the same dataset, there are four classes of syntactic differences between the queries resulting in situations where *DMQ* can be answered efficiently using the materialized results of *DMQ_V* since they correspond to equivalence, inclusion, and dominance relationships between *DMQ_V* and *DMQ*. These cases are listed below:

1. If *DMQ_V* and *DMQ* have the same pattern and time constraints, then the results of *DMQ* are equal to the results of *DMQ_V* (equivalence);
2. If *DMQ_V* and *DMQ* have the same time constraints, and *DMQ* extends pattern constraints of *DMQ_V*, then *DMQ* can be answered by filtering out the patterns returned by *DMQ_V* not satisfying pattern constraints of *DMQ* (inclusion according to the Theorem 1);
3. If *DMQ_V* and *DMQ* have the same pattern constraints, and *DMQ* extends time constraints of *DMQ_V*, then *DMQ* can be answered by evaluating the support of the patterns returned by *DMQ_V* using the time constraints of *DMQ*, and filtering out patterns not satisfying the minimum support threshold of *DMQ*. (dominance according to the Theorem 2);
4. If *DMQ* extends both pattern and time constraints of *DMQ_V*, then *DMQ* can be answered by evaluating the support of the patterns returned by *DMQ_V* using the time constraints of *DMQ*, and filtering out patterns not satisfying the pattern constraints of *DMQ*. (dominance according to the Theorem 3).

In all the cases we assume that the results of the user-specified query are to be written to disk. Answering the query in the case of equivalence is trivial (the results are already known and stored on the disk), therefore we concentrate on details concerning inclusion and dominance relationships.

In all algorithms presented below, *DMQ* represents the sequential pattern query issued by a user, *DMQ_V* is a query whose results are stored on disk, and *D* is the source dataset (a collection of data-sequences). Analyzing the cost (in terms of execution time) of the proposed algorithms, we assume that the following values are known: the number of data-sequences in the dataset (r_D) and the number of patterns in materialized query results (r_P); the number of disk blocks occupied by the dataset (b_D) and materialized query results (b_P). The following costs of disk and memory operations also appear in our cost formulas: the average cost of access to a disk block (c_D); the cost of checking if a pattern support exceeds a given threshold (c_S); the average cost of checking if a pattern satisfies given pattern constraints (c_F); the average cost of checking if a pattern is contained in a given data-sequence (c_C). For simplicity's sake, we treat checking pattern constraints and the containment test as elementary operations and use their average costs. We do not expect exact values of the last four parameters to be known. However, we believe that certain assumptions can made e.g. c_D is significantly larger than c_S, c_F, and c_C, c_C is greater than c_S, etc.

The overall cost of a given algorithm includes the cost of disk operations (reading the materialized patterns and the source dataset as well as writing the results) and computations in main memory. Operations in main memory concern individual data-sequences and patterns. The atomic portion of data read from or written to the disk is one disk block, which usually contains a number of patterns or data-sequences.

For the second case (inclusion due to extending pattern constraints) we propose an algorithm that performs one sequential scan of the materialized patterns, processing one pattern at a time. Each pattern is tested if it satisfies these basic Boolean pattern predicates from the pattern constraints of *DMQ* that were not in DMQ_V. All the basic Boolean pattern predicates of *DMQ* that were in DMQ_V must be satisfied by all the materialized patterns since pattern constraints in our model have the form of a conjunction of basic predicates. The algorithm for the second case is presented below.

Algorithm 1 (Result **F**iltering)

```
Answer = results of $DMQ_V$;
for each $p \in$ results of $DMQ_V$ do
begin
  for each basic Boolean pattern predicate $b$ such that
    $b$ is in pattern constraints of $DMQ$ and
    $b$ is not in pattern constraints of $DMQ_V$ do
      if not ($p$ satisfies $b$) then
        Answer = Answer \ {$p$};
        break;
      end if;
end;
output Answer;
```

The cost of Algorithm 1 can be expressed by the following formula (b_P" is the number of disk blocks to be occupied by the results of the query being answered):

$$C_F = b_P * c_D + r_P * c_F + b_P" * c_D \qquad (1)$$

Algorithms for the third and fourth cases (both leading to the dominance relationship) have to scan the source dataset once in order to re-evaluate the support of materialized patterns. In the third case, all that has to be done after the support re-evaluation is checking if new support values exceed the minimum support threshold of *DMQ*. The algorithm for the third case is presented below.

Algorithm 2 (Result **V**erification)

```
Answer = results of $DMQ_V$;
scan D once evaluating the support of patterns
in Answer using time constraints of DMQ;
for each $p \in$ Answer do
  if p exceeds the minimum support threshold of DMQ
  then output p; end if;
```

The cost of Algorithm 2 can be expressed by the following formula:

$$C_V = (b_P + b_D) * c_D + r_D * r_P * c_C + r_P * c_S + b_P'' * c_D \qquad (2)$$

The above formula is based on the assumption that the set of materialized patterns fits into main memory. Thus, the collection of materialized patterns is read only once. It should be noted that we do not make similar assumptions regarding the size of the source dataset. While scanning the source dataset, data-sequences can be read and processed one at a time.

For the fourth case (domination due to extending time and pattern constraints) we propose two algorithms. Algorithm 3, before scanning the source dataset, filters out patterns that do not satisfy pattern constraints of *DMQ* (including the minimum support threshold) using Algorithm 1. After the scan of the dataset, the minimum support threshold is checked again (it is the only one of predicate types that for a given pattern could by true before the support re-evaluation, and false after that operation). Algorithm 4 is a straightforward solution. It first re-evaluates the support of patterns being the results of *DMQ*$_V$, and then filters out patterns that do not satisfy pattern constraints of *DMQ*. The two algorithms exploiting results of a dominating query extending pattern constraints of the query to be answered are presented below:

Algorithm 3 (Result **F**iltering and **V**erification)

```
Answer = patterns in results of $DMQ_V$ satisfying
         pattern constraints of DMQ; /* Algorithm 1 */
scan D once evaluating the support of patterns
in Answer using time constraints of DMQ;
for each $p \in$ Answer do
  if p exceeds the minimum support threshold of DMQ
  then output p; end if;
```

The cost of Algorithm 3 can be expressed by the following formula (r_P' is the number of patterns whose support has to be re-evaluated during the dataset scan):

$$C_{FV} = (b_P + b_D) * c_D + r_P * c_F + r_D * r_P' * c_C + r_P' * c_S + b_P'' * c_D \qquad (3)$$

Algorithm 4 (Result **V**erification and **F**iltering)

```
Answer = results of $DMQ_V$;
scan D once evaluating the support of patterns
in Answer using time constraints of DMQ;
for each $p \in$ Answer do
  if p satisfies pattern constraints of DMQ
  then output p; end if;
```

The cost of Algorithm 4 can be expressed by the following formula:

$$C_{VF} = (b_P + b_D) * c_D + r_D * r_P * c_C + r_P * c_F + b_P'' * c_D \quad (4)$$

The advantage of the Algorithm 3 over the Algorithm 4 is the possible reduction of the number of patterns whose support has to be re-evaluated. However, Algorithm 3 compares supports of some patterns with the minimum support threshold twice (before and after the support re-evaluation). Let us evaluate the difference between the cost of applying the Algorithms 3 and 4 for the same query and exploiting the same materialized results of a previous dominating query:

$$C_{FV} - C_{VF} = r_D * r_P' * c_C + r_P' * c_S - r_D * r_P * c_C \quad (5)$$

If we assume that the Algorithm 3 during its initial pattern filtering phase filters out *x* patterns ($r_P - r_P' = x$), then we have:

$$C_{FV} - C_{VF} = (r_P - x) * c_S - r_D * x * c_C \quad (6)$$

Assuming that the number of materialized patterns used is smaller than the number of data-sequences in the source dataset ($r_P < r_D$), and the cost of pattern containment test is larger than the cost of comparing pattern's support with the minimum support threshold ($c_C > c_S$), the above formula states that if the initial filtering phase of the Algorithm 3 filters out at least one pattern ($x > 0$), then the Algorithm 3 outperforms the Algorithm 4. Thus, generating possible execution plans we do not consider application of the Algorithm 4 at all.

5 Cost-Based Optimization of Sequential Pattern Queries in Presence of Materialized Results of Previous Queries

A cost-based query optimizer works as follows. First, it generates all possible query execution plans. Next, the cost of each plan is estimated. Finally, based on the estimation, the plan with the lowest estimated cost is chosen. Since the decision is made using estimated cost values, the plan chosen may actually not be optimal. The quality of optimizer decisions depends on the complexity and accuracy of cost functions used. Cost functions that are used in practice usually do not take into consideration all the factors contributing to the execution costs. Cost estimation procedures have to be relatively simple, so that the time spent on optimization does not contribute significantly to the overall processing time.

We observe that the cost formulas that we provided in the previous section are not appropriate for the cost-based optimization for two major reasons. Firstly, the costs of disk and memory operations depend on a particular machine configuration. Secondly, the number of patterns after initial filtration in the cost formula for the Algorithm 3 is not known a priori. While the ratio between cost of disk and memory operations can be easily determined, estimating the size of materialized pattern collection after filtration according to a given pattern predicate is not trivial. The actual problem is that the selectivity of a given pattern predicate depends not only on the predicate itself but also on the patterns being filtered. To be able to estimate the selectivity of a given predicate on a given collection of patterns, the system could apply the predicate to a random sample of patterns. However, this would clearly result in more time spent on optimization, which is not desirable. Taking all this into account, we propose to use simplified cost functions expressed in terms of the number of disk blocks accessed. Since for a given query all correct execution plans have to lead to the same set of resulting patterns, in the simplified cost formulas we omit the blocks written while storing the results on disk. Thus, the new cost functions for the Algorithm 1 (F), Algorithm 2 (V), and Algorithm 3 (FV) look as follows:

$$C_F = b_P \qquad C_V = b_P + b_D \qquad C_{FV} = b_P + b_D \qquad (7)$$

Our optimizer performs the cost analysis only if there is more than one previous query including or dominating the current query, and no equivalent query can be found among the queries whose materialized results are available. The number of considered execution plans is equal to the number of including and dominating queries (Algorithm 1 is used for including queries, Algorithm 2 for dominating queries having the same pattern constraints, and Algorithm 3 for dominating queries extending pattern constraints of the current query). If the optimizer finds more than one including or dominating query, the cost of each execution plan has to be estimated using the cost formulas presented above. Then, the query is answered according to the plan with the lowest estimated cost. Of course, if no equivalent, including, or dominating query can be found, a complete sequential pattern mining algorithm has to be run.

6 Experimental Results

To support and verify our theoretical analysis, we performed several experiments on synthetic datasets generated by means of the *GEN* generator from the *Quest* project [1]. The size of the source dataset used in our experiments ranged from 1000 to 100000 data-sequences. For all datasets the total number of different items was 1000, average size of a data-sequence was 8 items, and the data distribution was the same. The time gap between two adjacent elements of each data-sequence was always equal to one time unit. The materialized collections of patterns contained from 800 to more than 6000 patterns. The source datasets and materialized query results were stored in a local *Oracle8i* database.

The first goal of the experiments was to verify whether disk activity is really a dominant contributor to the cost (expressed in terms of execution time) of the algorithms exploiting materialized patterns. For Algorithm 1, time spent on disk operations was on average 97% of the total time. However, in case of Algorithms 2 and 3 the average value of the above ratio was 78% and 92% respectively. The percentage of time spent on disk operations in case of Algorithm 3 varied significantly (from 75% to 98%) with the selectivity of pattern constraints used in queries. The experimental results prove that simplifying the cost formulas to the number of disk blocks accessed was justified. Nevertheless, it can lead to underestimating the cost of execution plans involving Algorithms 2 and 3.

The second goal was to check if the execution strategy chosen as the one resulting in the smallest number of disk accesses is really the optimal one. In general, minimizing disk accesses led to optimal execution plans. However, when we provided two materialized collections of patterns slightly differing in the number of occupied disk blocks, minimizing disk activity sometimes resulted in a non-optimal plan if at least one of the materialized collections of patterns required the application of the Algorithm 2 or 3. This is no surprise, since in case of those algorithms, main memory computations (not consider by simplified cost functions) are much more significant than in case of Algorithm 1. Nevertheless, the actual cost of the chosen strategy was never higher than the cost of the actual optimal strategy by more than 30%, which seems to be acceptable.

7 Conclusions

We addressed the problem of efficient answering sequential pattern queries in the presence of materialized results of previous sequential pattern queries. We focused on estimating the cost of execution strategies involving materialized collection of patterns. By comparing estimated costs of all execution strategies available for a given sequential pattern query, the data mining system can choose the execution plan that is optimal or close to optimal.

References

1. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd KDD Conference (1996)
2. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. 11th ICDE Conf. (1995)
3. Baralis E., Psaila G.: Incremental Refinement of Mining Queries. Proc. of the 1st DaWaK Conference (1999)
4. Elmasri R., Navathe S.B.: Fundamentals of Database Systems, Second Edition (1994)
5. Han J., Lakshmanan L., Ng R.: Constraint-Based Multidimensional Data Mining. IEEE Computer, Vol. 32, No. 8 (1999)
6. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
7. 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)
8. Oracle9i Database Performance Guide and Reference. Oracle Corporation (2001)
9. Parthasarathy S., Zaki M.J., Ogihara M., Dwarkadas S.: Incremental and Interactive Sequence Mining. Proc. of the 8th CIKM Conference (1999)
10. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)
11. Wojciechowski M.: Interactive Constraint-Based Sequential Pattern Mining. Proc. of the 5th ADBIS Conference (2001)