

A Systematic Approach for Optimizing Complex Mining Tasks on Multiple Databases

Ruoming Jin Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{jinr, agrawal}@cse.ohio-state.edu

Abstract

It has been well recognized that data mining is an interactive and iterative process. In order to support this process, one of the long-term goals of data mining research has been to build a Knowledge Discovery and Data Mining System (KDDMS). Along this line, much research has been done to provide database support for mining operations.

However, the focus in these efforts has typically been on mining a single dataset. In many situations, such as in a data warehouse, the user usually has a view of multiple datasets collected from different data sources. In such scenarios, comparing the patterns from different datasets and understanding their relationships can be an extremely important part of the KDD process. This requires support for complex queries on multiple datasets in a KDDMS.

This paper addresses the new functionality and optimizations required for the above process, specifically focusing on frequent itemset mining. We make the following contributions: 1) We present an SQL-based mechanism for querying frequent itemsets across multiple datasets. 2) We establish an algebra for such queries. 3) We develop a number of transformations on this algebra, and introduce several new operators for this purpose. 4) We present several algorithms for finding the query plan to reduce the execution cost. 5) We evaluate our algorithms on real and synthetic datasets, and show up to an order of magnitude performance gain.

1 Introduction

Within the last decade, data mining has emerged as an important component of databases and information systems. A large body of research exists on algorithms for a variety of data mining tasks, targeting a variety of applications, data types, and execution environments.

It has been well recognized that data mining is an interactive and iterative process, i.e., a data miner cannot expect to get interesting patterns and knowledge by a single execution of one algorithm. In order to support this process, one of the long-term goals of data mining research has been to build a *Knowledge Discovery and Data Mining System* (KDDMS) [10, 16, 18]. The vision is that such a system will

provide an integrated and user-friendly environment for efficient execution of data mining tasks or queries. Along this line, much research has been conducted to provide database support for mining operations. This includes the work on query language extensions [14, 17, 24] and implementing mining algorithms in a database system [8, 31, 37]. Logic and algebra based methods have also been proposed to model the mining process [7, 13, 19]. The subfield of constraint association mining allows mining of *interesting* association rules by taking of a variety of constraint conditions as input [6, 20, 22, 25, 28, 32].

In the above research projects, the focus has typically been on mining a *single* dataset. However, in many situations, such as in a data warehouse, the user usually has a view of multiple datasets collected from different data sources. In such scenarios, comparing the patterns from different datasets and understanding their relationships can be an extremely important part of the KDD process. This, however, requires support for complex queries on multiple datasets in a KDDMS.

Such support involves new optimizations as well as new functionality. Suppose a user needs to find itemsets that frequent with a certain support in both A and B . While this can be answered by taking intersection of the results from both A and B , this is likely to be very expensive. Instead, we can compute itemsets frequent in either of the two datasets, and then simply find which of these are frequent in the other dataset. However, this leads to two different evaluation plans, corresponding to using the dataset A and B , respectively, for the initial evaluation. The two evaluation plans can have different costs, depending upon the nature of the datasets A and B . Furthermore, as the number of datasets and the complexity of the query condition increases, the number of possible evaluation plans can also grow. Thus, there is a need for techniques for enumerating different query plans and choosing the one with the least cost, similar to what have been developed for relational queries.

Moreover, simply combining results from applying mining operators on each dataset may not even provide the results we desire. For example, suppose a user needs to find itemsets that are frequent with a certain support in either of the datasets A

or B , along with their frequencies in both these datasets. This requires more than just computing frequent itemsets in both A and B and taking their union. This is because we also need to determine the frequency of itemsets in a dataset which are only frequent in the other dataset.

This paper addresses the above two problems. We provide a simple mechanism for specifying mining queries across multiple datasets. Then, by representing these queries through an algebra, and developing a set of transformation and optimization techniques, we establish an approach for optimizing these queries. Our work is specifically in the context of frequent itemset mining, which is a core data mining operation. Algorithms for frequent itemset mining have formed the basis for a number of other mining problems, including association mining, correlations mining, and mining sequential and emerging patterns [15].

To summarize, this paper makes the following contributions:

- We present an SQL based mechanism for querying frequent itemsets across multiple datasets.
- We establish an algebra for such queries.
- We introduce several new operators and develop a number of transformations on this algebra to enable aggressive optimizations.
- We present several heuristic algorithms for finding efficient query plans.
- We evaluate our query optimization techniques on both real and synthetic datasets, and demonstrate up to an order of magnitude performance gains as compared to the naive execution.

The rest of the paper is organized as follows. In Section 2, we describe several scenarios in which queries over multiple datasets can arise. In Section 3, our query framework is presented. In Section 4, we establish an algebra for the class of queries we are interested in. An overview of the query optimization is presented in Section 5. Details of transformations and query plan generation are considered in Section 6. Our experimental evaluation is presented in Section 7. We compare our work with related research efforts in Section 8 and conclude in Section 9.

2 Motivating Examples

To further motivate and facilitate our study, we consider two scenarios and list many examples of the kind of queries our framework targets.

Mining the Data Warehouse for a Nation-wide Store: Consider a store that has three branches, in New Jersey, New York, and California, respectively. Each of them maintains a database with last one week’s retail transactions. To understand how the geographical factors impact shopping patterns, queries of the following type are likely to be asked:

- Q1: Find the itemsets that are frequent with support level 0.1% in *any* of the stores.
- Q2: Find the itemsets that are frequent with support level 0.1% in *each* store.

- Q3: Find the itemsets that are frequent with support level 0.05% in both the stores on east coast, but are very infrequent (support less than 0.01%) in the west coast store.

Finding Signature Itemsets for Network Intrusion: In a signature detection system, frequent itemsets can serve as the patterns to signal well-known attacks [26]. Suppose a *tcp-dump* dataset contains the TCP packet information of several different network intrusion attacks. We can split the available data into several datasets, with one dataset corresponding to each intrusion type and a *normal* dataset corresponding to the situation when no intrusion is occurring. Queries of the following type have been used to capture the signature patterns [26]:

- Q4: Find the itemsets that are frequent with a support level 80% in *either* of the intrusion datasets, but are very infrequent (support less than 50%) in the normal dataset.
- Q5: Find the itemsets that are frequent with a support level 70% in *each* of the intrusion datasets, but are very infrequent (support less than 60%) in the normal dataset.
- Q6: Find the itemsets that are frequent with a support level 85% in *one* of the intrusion datasets, but are very infrequent (support less than 65%) in all other datasets.

In this paper, we have the following two goals. First, we want to have a systematic and simple way of expressing the above queries. Second, we want to be able to optimize the above queries.

3 SQL Extensions for Mining Across Multiple Datasets

As we stated earlier, many research projects have proposed database language extensions for expressing mining operations, including finding frequent itemsets. However, none of these provide convenient basis for expressing and optimizing frequent itemset computations across multiple datasets. In this section, we introduce a new approach to querying frequent itemsets across multiple datasets. Our approach involves using a virtual table, and using SQL queries to partially materialize the virtual table.

Let $\{A_1, A_2, \dots, A_m\}$ be the set of datasets we are targeting. Each of these comprises transactions, which are set of items. The datasets are also *homogeneous*, i.e., an item has an identical name across different datasets. Let *Item* be the set of all the possible items in all datasets.

We define the following schema,

$$Frequency(I, A_1, A_2, \dots, A_m)$$

For a table F of this schema, the column with attribute $F.I$ stores all possible itemsets, i.e., the power-set of *Item*. The column with attribute $F.A_i$ stores the frequency of the itemsets in the dataset A_i . For example, consider two transaction datasets A_1 and A_2 , as shown in Table 1. The set of distinct items in the two datasets, *Item*, is $\{A, B, C, D, E\}$. Table 2 contains a portion of the F table for the datasets A_1 and A_2 .

Dataset A_1		Dataset A_2	
TransID	Items	TransID	Items
1	{A,B,E}	1	{A B D E}
2	{B,D}	2	{B C E}
3	{A, B, E}	3	{A, B, E}
4	{A,C, D}	4	{A, B, C}
5	{B,C,D}	5	{A, C}
6	{A,C,D}	6	{C, D}
7	{A, B}		
8	{A, B, C, D, E}		

Table 1. Datasets A_1 and A_2

I	A_1	A_2
{A}	6/8	4/6
{B}	6/8	4/6
{C}	4/8	4/6
{D}	6/8	2/6
{E}	3/8	3/6
{A,B}	4/8	3/6
{A,C}	3/8	2/6
⋮	⋮	⋮
{A,B,C,D,E}	1/8	0

Table 2. F Table for the Datasets A_1 and A_2

In real scenarios where frequent itemset mining is applied, the number of distinct itemsets can easily be in hundreds, thousands, or even more. Consequently, the total number of itemsets is likely to be too large for the table F to be materialized and stored. Thus, such a table can only be used as a virtual table or a logical view.

With this, a query to partially materialize the virtual frequency table F has following format:

```
SELECT  $F.I, F.A_{i_1}, F.A_{i_2}, \dots, F.A_{i_s}$ 
FROM  $Frequency(I, A_1, \dots, A_m)$   $F$ 
WHERE  $Condition C$ 
```

where, $\{A_{i_1}, \dots, A_{i_s}\} \subseteq \{A_1, \dots, A_m\}$ and a condition is defined as follows:

- $F.A_i \geq \alpha$ and $F.A_i < \alpha$ are conditions, where $0 \leq \alpha \leq 1$.
- $(C_1 \text{ AND } C_2)$ and $(C_1 \text{ OR } C_2)$ are conditions if C_1 and C_2 are conditions.
- $(NOT(C))$ is a condition if C is a condition.

The above query format and the set of allowable conditions can be used to express a rich class of queries. However, the number of infrequent itemsets in a single dataset is usually much larger than the number of frequent itemsets, and enumerating all infrequent itemsets and computing their frequencies can be extremely expensive. To avoid generating such a large number of infrequent itemsets, we require that each query involving infrequent itemsets on a given dataset must be constrained by at least one query involving frequent itemsets. This is formalized as the *admissible condition* on a query.

Formally, let $F.A_j \geq \alpha$ be considered a positive predicate and $F.A_i < \alpha$ be considered a negative predicate. A condition is mapped to a logical formula, which is then transformed into the following disjunctive normal form (DNF).

$$C = C_1 \vee C_2 \vee \dots \vee C_k$$

where, C_i is a *conjunctive-clause*, i.e., it involves AND operation on one or more predicates.

With this, we define the following:

Definition 1 A *conjunctive-clause* is constrained if it includes a positive predicate.

Definition 2 A *condition* is admissible if every conjunctive-clause in its DNF format is constrained.

For example, the following condition to query the F table on datasets A_1, A_2, A_3 is **not** admissible.

$$F.A_1 < 0.1 \text{ OR } (F.A_2 \geq 0.2 \text{ AND } F.A_3 < 0.05)$$

This is because the first conjunctive-clause, $F.A_1 < 0.1$, is not constrained.

In the rest of this paper, we will only focus on queries with admissible conditions.

4 Algebra for Queries

In the previous section, we introduced a query format for expressing frequent itemset operations on multiple datasets. This section develops an algebra formulation for processing these queries.

We begin with the definition of a *view* of the F table. A view of the F table is a table with a subset of the rows and columns of the F table, which always contains the column of the attributes I , and the exact frequency of an itemset can be replaced by a *Null* value (denoted as \circ).

Given this, we define three basic operators, which can generate simple views of the F table.

1. The frequent itemset mining operator $SF(A_j, \alpha)$ computes the frequent itemset from a single dataset A_j with support level α . It returns a set of two-tuples, comprising itemsets and their frequency on dataset A_j .

2. The negative frequent itemset mining operator $SF^-(A_j, \alpha)$ computes itemsets in A_j with support level less than α . Formally, assuming 2^{Item} to be the power-set of $Item$, and $SF^I(A_j, \alpha)$ is the projection of $SF(A_j, \alpha)$ on the column of attributes I , we have

$$\overline{SF(A_j, \alpha)} = (2^{Item} - SF^I(A_j, \alpha)) \times \{\circ\}$$

3. The counting operator $P(X, A_j)$ counts the frequency for each itemset in the set X on dataset A_j . To simplify its evaluation, this operator is only defined on a set X that satisfies the *down-closure* property, i.e, if an itemset is in X , then all of its subsets are also in X .

Note that each operator discussed above generates simple views of the F table. Table 3 shows some examples of the basic operators, where the datasets A_1 and A_2 are as shown in Table 1, and X is $\{\{A\}, \{B\}, \{E\}, \{A, B\}, \{A, E\}, \{B, E\}, \{A, B, E\}\}$.

Next, we define two operations that can combine the views of the the F table. Let F_1 and F_2 be two views of the F table. Let F_1^I and F_2^I be the projections of F_1 and F_2 on the attribute I . We define the following two operations:

1. Intersection (\cap): $F_1 \cap F_2$ is defined as

$$(F_1^I \cap F_2^I) \bowtie_I F_1 \bowtie_I F_2$$

$SF(A_1, 0.5)$		$SF(A_2, 0.4)$		$SF(A_2, 0.4)$		$P(X, A_1)$	
I	A_1	I	A_2	I	A_2	I	A_1
{A}	6/8	{A}	4/6	{D}	○	{A}	6/8
{B}	6/8	{B}	4/6	{A,C}	○	{B}	6/8
{C}	4/8	{C}	4/6	{A,D}	○	{E}	3/8
{D}	6/8	{E}	3/6	{A,E}		{A,B}	4/8
{A,B}	4/8	{A,B}	3/6	:	:	{A,E}	3/8
{C,D}	4/8			{A,B,C, D,E}	○	{B,E}	3/8
						{A,B,E}	3/8

Table 3. Basic Operators on F Table

$SF(A_1, 0.5) \sqcap SF(A_2, 0.4)$			$SF(A_1, 0.5) \sqcup SF(A_2, 0.4)$		
I	A_1	A_2	I	A_1	A_2
{A}	6/8	4/6	{A}	6/8	4/6
{B}	6/8	4/6	{B}	6/8	4/6
{C}	4/8	4/6	{C}	4/8	4/6
{A,B}	4/8	3/6	{D}	6/8	○
			{E}	○	3/6
			{A,B}	4/8	3/6
			{C,D}	4/8	○

Table 4. Intersection and Union Operation

In this operation, we find the itemsets that are common to F_1 and F_2 , i.e. the set $(F_1^I \cap F_2^I)$. We report the frequency count of such itemsets for all datasets included in either F_1 or F_2 . This is done by essentially taking a join over the attribute I of $(F_1^I \cap F_2^I)$ with F_1 and F_2 . \bowtie is the standard database join operation, with one important difference. Any column that is common between F_1 and F_2 is *merged*. In merging the columns, an actual count is preferred over a \circ (Null) value.

2. Union (\sqcup): $F_1 \sqcup F_2$ is defined as

$$(F_1^I \cup F_2^I) \bowtie_I F_1 \bowtie_I F_2$$

In this operation, we find the itemsets that are in either F_1 or F_2 , i.e. the set $(F_1^I \cup F_2^I)$. We now take an *outerjoin* [34]. We report frequency count for these itemsets and all datasets that are in F_1 or F_2 . Null is inserted for entries for which values are not available from either F_1 or F_2 .

Note that the results of the two operations are still views of the F table. Table 4 provides examples for each of these two operations.

Based upon the definitions of the above operations, we can easily prove the following:

Lemma 1 *The operations, intersection (\sqcap) and union (\sqcup), satisfy the associative, commutative, and distributive properties.*

Next, we discuss how a query can be modeled using the above operators and operations. Let us consider a query Q with the condition C . As stated earlier, the condition C can be restated in the DNF form, with conjunctive clauses C_1, \dots, C_k . Formally,

$$C = C_1 \vee \dots \vee C_k, \quad C_i = C_{i1} \wedge \dots \wedge C_{im}, \quad 1 \leq i \leq k$$

If C_{il} is a positive predicate, we can replace it by the operator $SF(A_j, \alpha)$. Similarly, we replace a negative predicate by

$\overline{SF(A_j, \alpha)}$. We can represent the query by

$$F_C = F_{C_1} \sqcup \dots \sqcup F_{C_k}$$

where, each F_{C_i} is computed using intersection operations.

Thus, we have captured a condition C using the operators and operations we have defined. One particular issue, however, still needs to be addressed. The table F_C can contain \circ (Null) values. An example of this is $SF(A_1, 0.5) \sqcup SF(A_2, 0.4)$ in the Table 4. To correctly evaluate the original query, the null value of an itemset needs to be replaced by its actual frequency. This can be done by applying the counting operator we have defined.

It turns out that a modified version of the query can better capture the intentions of the user and avoid the costs associated with counting operators. A null value of an itemset suggests that the itemset is *infrequent* with respect to the specified support level. In such cases, a user could simply be interested in knowing that the itemset is infrequent in the particular dataset, and may not need to know the specific frequency value. We introduce a new notation, g , for this purpose. In the Select clause of original query, replacing A_i by $g(A_i)$ denotes that the null value is acceptable if the itemset is infrequent. The significance of $g(A_i)$ for our query evaluation is that the counting operator on A_i is not required.

Given this, the queries can be generalized as follows:

```
SELECT F.I, [F.Ai1|g(F.Ai1)], ..., [F.Ais|g(F.Ais)]
FROM Frequency(I, A1, ..., Am) F
WHERE Condition C
```

where, $\{A_{i1}, \dots, A_{is}\} \subseteq \{A_1, \dots, A_m\}$.

For such a general query Q , whose condition C is captured using the expression F_C , the result can be expressed as

$$R(Q) = F_C \bowtie_I P(\widehat{F}_C^I, A_{i1}) \bowtie_I \dots \bowtie_I P(\widehat{F}_C^I, A_{it})$$

where, $\{A_{i1}, A_{i2}, \dots, A_{it}\}$, denoted as $CSET$, is the set of datasets appearing in the SELECT clause without the g notation, and \widehat{F}_C^I is the minimal extension of F_C^I which satisfies the down-closure property.

5 Query Optimization Overview

This section gives an overview of the issues involved in correctly and efficiently evaluating a query of the form defined in the previous section.

To facilitate our discussion, we use the following query, denoted by Q , as a running example.

```
SELECT F.I, g(F.A), g(F.B), F.C, g(F.D)
FROM Frequency(I, A, B, C, D) F
WHERE (F.A ≥ 0.1 AND F.B ≥ 0.1 AND
NOT (F.C ≥ 0.05 OR F.D ≥ 0.05))
OR (F.C ≥ 0.1 AND F.D ≥ 0.1 AND
NOT (F.A ≥ 0.05 OR F.B ≥ 0.05))
```

The query involves finding the itemsets which are frequent with support level 0.1 in both the datasets A and B , but infrequent (support less than 0.05) in the datasets C and D , or vice versa. The DNF form of the condition C is:

$$(A \geq 0.1 \wedge B \geq 0.1 \wedge C < 0.05 \wedge D < 0.05) \vee (C \geq 0.1 \wedge D \geq 0.1 \wedge A < 0.05 \wedge B < 0.05)$$

F_C can be expressed as:

$$(SF(A, 0.1) \sqcap SF(B, 0.1) \sqcap \overline{SF(C, 0.05)} \sqcap \overline{SF(D, 0.05)}) \sqcup (SF(C, 0.1) \sqcap SF(D, 0.1) \sqcap \overline{SF(A, 0.05)} \sqcap \overline{SF(B, 0.05)})$$

The answering set of this query can be expressed as

$$R(Q) = F_C \bowtie_I P(\widehat{F_C^I}, C)$$

As suggested in the expression $R(Q)$, the naive method to evaluate the query Q , needs to invoke the SF operator 8 times and the counting operator P once. Note that the negative frequent itemset mining operator, \overline{SF} , is actually implemented by frequent itemset mining operator, SF , and its result view does not need to be materialized in order to evaluate the query. The reason for this is as follows. Recall that every conjunctive-clause in the DNF format of the admissible condition contains at least one positive predicate. Therefore, in F_C , each negative frequent itemset mining operator, $\overline{SF}(A_i, \alpha_i)$, will intersect (\sqcap) with some frequent itemset mining operator, $SF(A_j, \alpha_j)$. By applying the property of the operation \sqcap , $\overline{SF}(A_i, \alpha_i)$ can be more efficiently evaluated as

$$(SF^I(A_j, \alpha_j) - SF(A_i, \alpha_i)^I) \times \{o\}$$

Now, let us consider the costs associated with evaluating the expression $R(Q)$, and the opportunities for optimizing it. The first observation is that the costs of SF and P operators are typically much higher than those of union and intersection operations. Therefore, we need to focus on SF and P operators in our optimization process.

Let us consider the naive evaluation of $R(Q)$. The key observation is that a large fraction of the computation is either *repetitive* or *unnecessary*. By *repetitive* computation, we imply finding the frequency of an itemset on a dataset more than once, due to different mining operators. For example, the computation of $SF(A, 0.1)$ is repetitive. This is because $SF(A, 0.05)$ is also evaluated and $SF(A, 0.1) \subseteq SF(A, 0.05)$. By *unnecessary* computation, we imply finding the frequency of the itemsets which do not appear in the result $R(Q)$. For example, the computation of frequency for each itemset in the set $SF^I(A, 0.1) - F_C^I$ on the dataset A is unnecessary.

In optimizing the query evaluation process, our first goal is to try and remove repetitive and unnecessary computations. This is done by introducing new operators and using *containing* relations, respectively. These are discussed in the next two subsections.

5.1 New Operators

To reduce the unnecessary computation, two new operators, CF and GF , are introduced.

1. Frequent itemset mining operator with constraints $CF(A_j, \alpha, X)$ finds the itemsets that are frequent in the dataset A_j with support α and also appears in the set X . X is a set of itemsets that satisfies the down-closure property. This operator also reports the frequency of these itemsets in A_j .

Formally, $CF(A_j, \alpha, X)$ computes the following view of the F table:

$$X \sqcap SF(A_j, \alpha)$$

The typical scenario where this operator helps remove unnecessary computation is as follows. Suppose the frequent itemset operator intersects with some view of the F table, such that the projection of this view on the attribute I is X . This operator pushes the set X into the frequent itemset generation procedure, i.e., X serves as the search space for the frequent itemset generation. Thus, the unnecessary computation for the itemsets that are not in X can be saved.

2. Group frequent itemset mining operator $GF(Y)$, where $Y = \{ \langle A_1, \alpha_1 \rangle, \dots, \langle A_u, \alpha_u \rangle \}$, finds the itemsets that are frequent in each dataset A_i with support α_i , and reports their frequency in each of these datasets. Formally, $GF(Y)$ computes the following view of the F table:

$$SF(A_1, \alpha_1) \sqcap \dots \sqcap SF(A_u, \alpha_u)$$

The basic idea behind this operator is as follows. The frequency count for all datasets in Y is carried out in parallel. Thus, all supersets of an itemset that is determined to be infrequent in any of the datasets is pruned.

We use the following example to illustrate the use of these operators. Consider the following view of the F table,

$$(SF(A, 0.1) \sqcap SF(B, 0.1)) \sqcup (SF(A, 0.1) \sqcap SF(C, 0.1))$$

Applying the CF operator, we can evaluate $SF(A, 0.1)$ first, and then intersect it with

$$(CF(B, 0.1, SF^I(A, 0.1)) \sqcup CF(C, 0.1, SF^I(A, 0.1)))$$

Compared with the naive method, this evaluation reduces the unnecessary costs of $SF^I(B, 0.1) - (SF(A, 0.1) \sqcap SF(B, 0.1))^I$ on the dataset B and $SF^I(C, 0.1) - (SF(A, 0.1) \sqcap SF(C, 0.1))^I$ on the dataset C . However, the computation for $SF^I(A, 0.1) - (SF(B, 0.1) \sqcup (SF(C, 0.1)))^I$ on the dataset A is still unnecessary.

Applying the GF operator, this view can be evaluated as

$$GF(\{ \langle A, 0.1 \rangle, \langle B, 0.1 \rangle \}) \sqcup GF(\{ \langle A, 0.1 \rangle, \langle C, 0.1 \rangle \})$$

No unnecessary computation is involved now. However, the computation of the itemsets in the set $(SF(A, 0.1) \sqcap SF(B, 0.1) \sqcap SF(C, 0.1))^I$ for dataset A has now become repetitive.

5.2 Containing Relation

An important tool to remove repetitive computation is based on the *containing relation* for the sets of frequent itemsets. The containing relation is as follows: $\beta \leq \alpha$, $SF(A_j, \beta)$ contains all the frequent itemsets in $SF(A_j, \alpha)$. Therefore, if the first one is available, invocation of the second can be avoided. Instead, a relatively inexpensive selection operator, denoted as σ , can be applied. Formally, for $\beta \leq \alpha$, we have,

$$SF(A_j, \alpha) = \sigma_{A_j \geq \alpha}(SF(A_j, \beta))$$

This containing relations can be also extended to the our two new operators, CF and GF .

Let us revisit our running example. In view of this relation, at most one invocation of the mining operator SF on each dataset is required. Thus, we only need four invocations of the SF operator, i.e., $SF(A, 0.05)$, $SF(B, 0.05)$, $SF(C, 0.05)$, and $SF(D, 0.05)$. This method, which removes all repetitive computation due to SF operator, but does not use CF and GF operators, is referred to as the *Optimization RR* (Remove Redundant). It should be noted that though the repetitive computation due to SF operator is removed here, much unnecessary computation is still involved.

5.3 Overview of Query Plan Generation

The discussion in the previous two subsections focused on removing unnecessary and repetitive computations, respectively. Each was considered independently. In generating an efficient plan for evaluating a query, it is important to consider both. As our example has shown, removing unnecessary computation can introduce repetitive computation, and vice-versa. Clearly, this makes query optimization a challenging task. In many cases, removing both unnecessary and repetitive computation for a query evaluation is not possible. Particularly, if negative frequent itemset mining operator and counting operator occur in the query, finding an efficient query plan can become very difficult.

In the next section, we present a systematic approach for finding efficient query plans. Our approach includes the following three steps.

1. **Transformations:** This step removes both the negative frequent itemset mining operator \overline{SF} and counting operator P from $R(Q)$. The format of $R(Q)$ without \overline{SF} and P is referred to as the *standard form* of $R(Q)$.
2. **M table Formulation:** The standard form of $R(Q)$ is encoded into an M table. In the M table, each column represents a conjunctive-clause in the condition, and each row represents a dataset. Each cell in the table contains a predicate that appears in the condition and needs to be evaluated. Further, the query evaluation process can be depicted as a coloring scheme of the M table.
3. **Query Plan Generation:** The efficient query plans are generated with the help of the coloring scheme of M table.

6 Query Transformation, Evaluation, and Optimization

This section presents our approach for query plan generation. The transformations are introduced in Subsection 6.1. The M table and its coloring scheme are discussed in Subsection 6.2. Finally, Subsection 6.3 presents several algorithms for generating efficient query plans.

6.1 Transformations for Query Optimization

In the following, we introduce two transformations which can remove the the negative frequent itemsets operator \overline{SF} and the counting operator P from $R(Q)$, and replace them by $F(A_j, \alpha)$ operators. As stated earlier, the transformed $R(Q)$ is referred to as the standard form.

Transformation 1: (Removing Counting Operator) This transformation takes three steps. In the first step, for any dataset $A_j \in CSET$, which suggests that a counting operator P might be needed, we add the boolean clause $A_j \geq 0$ into every conjunctive-clause in the DNF format of condition C . Thus, we generate a new condition, denoted as C' . Clearly, in this new condition, two boolean clauses on the same dataset may appear in a single conjunctive-clause. In the second step, we remove these redundant boolean clauses by the following rule. If the boolean clause besides the new one is positive, the new one is dropped, and if the boolean clause besides the new one is negative, the negative boolean clause is dropped. Finally, we construct $F_{C'}$ corresponding to condition C' after the second step, and apply the selection operator with condition C to get $R(Q)$. Formally,

$$R(Q) = \sigma_C(F_{C'})$$

Let us illustrate this transformation on our running example. The set $CSET$ includes only the dataset C . In the first step, the new condition C' is

$$(A \geq 0.1 \wedge B \geq 0.1 \wedge C < 0.05 \wedge D < 0.05 \wedge C \geq 0) \vee (C \geq 0.1 \wedge D \geq 0.1 \wedge A < 0.05 \wedge B < 0.05 \wedge C \geq 0)$$

In the second step, the condition C' becomes:

$$(A \geq 0.1 \wedge B \geq 0.1 \wedge C \geq 0 \wedge D < 0.05) \vee (C \geq 0.1 \wedge D \geq 0.1 \wedge A < 0.05 \wedge B < 0.05)$$

In the final step, we construct $F_{C'}$,

$$F_{C'} = (SF(A, 0.1) \sqcap SF(B, 0.1) \sqcap SF(C, 0) \sqcap \overline{SF(D, 0.05)}) \sqcup (SF(C, 0.1) \sqcap SF(D, 0.1) \sqcap \overline{SF(A, 0.05)} \sqcap \overline{SF(B, 0.05)})$$

The answering set $R(Q)$ becomes $\sigma_C(F_{C'})$.

Transformation 2: (Removing Negative Frequent Itemset Operator) This transformation is based upon the following Lemma.

Lemma 2 *Let C be any condition, and F_C is the set satisfying this condition, then we have*

$$F_C \sqcap \overline{SF(A_j, \alpha)} = \sigma_{C \wedge (A_j < \alpha)}(F_C \sqcup (F_C \sqcap SF(A_j, \alpha)))$$

Note that the $NULL(\circ)$ value is treated as 0. The detailed proof is omitted here, but the correctness of this lemma can be observed from the fact that

$$F_C \sqcap \overline{SF(A_j, \alpha)} \subseteq F_C \sqcup (F_C \sqcap SF(A_j, \alpha))$$

This lemma suggests that the negative frequent itemset operator can be removed by applying the *union* (\sqcup), *intersection* (\sqcap), and selection operator.

By applying Lemma 2, all the negative frequent itemset operator can be dropped from $F_{C'}$. Let

$$F_{C_j} = SF(A_{i1}, \alpha_1) \sqcap \cdots \sqcap SF(A_{iu}, \alpha_u) \sqcap \overline{SF(A_{i(u+1)}, \alpha_{u+1})} \sqcap \cdots \sqcap \overline{SF(A_{is}, \alpha_s)}$$

	F_1	F_2	F_3	F_4	F_5
A	0.1	0.1		0.05	
B	0.1	0.1			0.05
C	0	0	0.1	0.1	0.1
D		0.05	0.1	0.1	0.1

Table 5. M Table for the query Q

We denote $F_{C_j^+}$ to contain only the sets of frequent itemsets for C_j , such as

$$F_{C_j^+} = SF(A_{i1}, \alpha_1) \sqcap \dots \sqcap SF(A_{iu}, \alpha_u)$$

Therefore, we have the following equality:

$$F_{C_j} = \sigma_{C_j}(F_{C_j^+} \sqcup (F_{C_j^+} \sqcap SF(A_{i(u+1)}, \alpha_{u+1})) \sqcup \dots \sqcup (F_{C_j^+} \sqcap SF(A_{is}, \alpha_s)))$$

Further, we can see that for each F_{C_j} , the selection operator (σ) can be dropped because of the outside selection operator. In sum, this transformation removes all the negative frequent itemset mining operator, such as $\overline{SF}(A_j, \alpha)$, in F_{C_j} by applying this equality and dropping the selection operator for each conjunctive clause C_j .

After these two transformations, the entire computation cost to evaluate the query Q has been shifted to compute $F_{C'}$. To simplify the discussion, we treat computing $F_{C'}$ as an instance of this generalized problem of evaluating expression F , where,

$$F = F_1 \sqcup \dots \sqcup F_t$$

and,

$$F_j = SF(A_{j1}, \alpha_{j1}) \sqcap \dots \sqcap SF(A_{jh}, \alpha_{jh})$$

Therefore, in our example, we have

$$\begin{aligned} F = F_{C'} &= (SF(A, 0.1) \sqcap SF(B, 0.1) \sqcap SF(C, 0)) \Rightarrow F_1 \\ &\sqcup (SF(A, 0.1) \sqcap SF(B, 0.1) \sqcap SF(C, 0) \sqcap SF(D, 0.05)) \Rightarrow F_2 \\ &\quad \sqcup (SF(C, 0.1) \sqcap SF(D, 0.1)) \Rightarrow F_3 \\ &\quad \sqcup (SF(C, 0.1) \sqcap SF(D, 0.1) \sqcap SF(A, 0.05)) \Rightarrow F_4 \\ &\quad \sqcup (SF(C, 0.1) \sqcap SF(D, 0.1) \sqcap SF(B, 0.05)) \Rightarrow F_5 \end{aligned}$$

6.2 A Unified Query Evaluation Scheme

This subsection describes a general scheme for query evaluation based on the standard form of answering set for queries.

As we had shown toward the end of previous subsection, after applying the two optimizations and two transformations we had listed, the standard query has the form $F = F_1 \sqcup \dots \sqcup F_t$. Let m be the number of distinct datasets that appear in F .

We can depict the query evaluation problem by using a table M with m rows and t columns. The row i in the table M corresponds to the dataset A_i , and the column j corresponds to the clause F_j . If $SF(A_i, \alpha)$ appears in F_j , the cell at j -th column and i -th row will have α , i.e., $M_{i,j} = \alpha$, otherwise, the cell $M_{i,j}$ is empty. In our example, the table has 4 rows and 5 columns, and is shown in Table 5.

We introduce the notion of *necessary information* for a non-empty cell. For a cell $M_{i,j}$, the necessary information is formally defined as

$$F_j^I \bowtie_I SF(A_i, M_{i,j})$$

Intuitively, this is the set that needs to be computed to evaluate this cell of the table.

Next, we capture the evaluation process for a query by using a simple coloring scheme for the cells in the table. Initially, all the cells are black. If the necessary information for a cell is available, or its superset is available, the cell is colored red. Clearly, the query evaluation process is complete when all non-empty cells are colored red.

To facilitate the discussion, we define a chain of sets on each row (i.e. for each dataset) of the M table. Suppose $\alpha_1^i > \dots > \alpha_u^i$ are the distinct support levels appearing in row i . We construct the sets $S_{i,r}$ to include the cells at row i that has a support larger than or equal to α_r^i . Formally, for each $r, 1 \leq r \leq u$,

$$S_{i,r} = \{j | M_{i,j} \geq \alpha_r^i, 1 \leq j \leq t\}$$

For example, the first row for the dataset A in the M table of the query Q has two support levels, $\alpha_1^1 = 0.1$ and $\alpha_2^1 = 0.05$, and the corresponding chain of sets is $S_{1,1} = \{1, 2\}$ and $S_{1,2} = \{1, 2, 4\}$.

It is easy to see that $S_{i,1} \subset \dots \subset S_{i,u}$. Thus, we have a chain based on the containing relation.

The query evaluation process proceeds in steps. In each step, one or more non-empty cells are colored red. At a stage p , suppose for any column j , the cells $M(k1, j), \dots, M(ku, j)$ are colored red, while the others are still black. Then, the information available at this stage can be denoted by:

$$F[p] = \sqcup_{1 \leq j \leq t} (\sqcap_{i \in \{k1, \dots, ku\}} SF(A_i, M(i, j)))$$

Next, we look at how the operators, $SF(A_i, \alpha)$, $CF(A_i, \alpha, X)$, and $GF(Y)$ can color the table. In each invocation of a mining operator with fixed dataset and support level, we consider how the set of frequent itemsets can maximally provide information for the cells, and hence color them. **Frequent mining operator $SF(A_i, \alpha)$:** Let α rank r in the sequence of support levels for the row i and have the corresponding set $S_{i,r}$. An invocation of the frequent mining operator on the dataset A_i , with support α , will turn each cell at row i whose column appears in the set $S_{i,r}$ red. The coloring that will occur is independent of the current coloring of the table M .

Frequent mining operator with constraint $CF(A_i, \alpha, X)$: Again, let α rank r in the sequence of support levels for the row i and have the corresponding set $S_{i,r}$. The coloring impacted by this operator is dependent on the current coloring of the table M . Let S be the set of columns which meet the following three conditions: 1) they are in the set $S_{i,r}$, 2) they have at least one cell colored red, and 3) their cell at row i is black. Let X be the set of frequent itemsets extracted from the columns in S . Then, by applying this operator on dataset A_i with support α and the set X , all cells on row i whose column appears in S will turn red.

Group frequent itemset mining operator $GF(Y)$: Invoking this operator is also independent of the coloring of the

	F_1	F_2	F_3	F_4	F_5
A	0.1	0.1		0.05	
B	0.1	0.1			0.05
C	0	0	0.1	0.1	0.1
D		0.05	0.1	0.1	0.1

Table 6. Colored M Table for the query \mathcal{Q}

table. Let $Y = \{ \langle A'_1, \alpha_1 \rangle, \dots, \langle A'_u, \alpha_u \rangle \}$. Let the dataset $A'_k, 1 \leq k \leq u$ correspond to the row ik . Let the support α_k rank jk at row ik , and hence correspond to the set $S_{ik,jk}$. Let $S = S_{i1,j1} \cap \dots \cap S_{iu,ju}$. Invoking this operator will turn every cell in the rectangle defined by $\{i1, \dots, iu\} \times S$ red.

Consider applying $SF(A, 0.05), CF(B, 0.1, SF^I(A, 0.1))$, and $GF(\{C, 0.1\}, \{D, 0.1\})$ consecutively on an initially black-colored table M of the query \mathcal{Q} . The first operator, SF , will turn the cells $M_{1,1}, M_{1,2}$, and $M_{1,4}$ red; the second operator, CF , will turn the cells $M_{2,1}$ and $M_{2,2}$ red; the third operator, GF , will turn the cells the right-bottom rectangle defined by $\{3, 4\} \times \{3, 4, 5\}$ red. Table 6 shows the resulting colored table.

By the above formulation, the query evaluation problem has been converted into the problem of coloring the table M . Different operators can be used, and in different order, to color the entire table red. There are different costs associated with each. The next subsection addresses the problem of finding efficient query evaluation plans.

6.3 New Query Plans

Using the table coloring formulation from the previous subsection, we now discuss approaches to finding an efficient query evaluation plan.

The key difficulty in this optimization process is that it is very hard to associate cost functions for the three operators. We are not aware any research on predicting the running time for a specific mining algorithm on a given dataset. The use of the operator CF that we have introduced further complicates this, because the performance can vary depending upon the status of the table. Therefore, we use a set of heuristics and greedy algorithms for this purpose. We first present two algorithms that are based upon the use of the SF and CF operators. Then, we describe another algorithm that further exploits the GF operator.

6.3.1 Using Constraint Based Operator

The constraint based mining operator $CF(A_j, \alpha, X)$ helps reduce the computational cost as follows. At any stage p , suppose that we need to color the cell $M_{i,j}$. As long as another red cell is available in the same column, CF operator can be used.

The algorithms we present here are based upon aggressively using the CF operator. In order to apply this operator, a query plan needs to be split into two phases:

- In the *first* phase, we use the $SF(A_j, \alpha)$ operators so that each column has at least one red cell.
- In the *second* phase, we use the $CF(A_j, \alpha, X)$ operators to compute all other non-empty cells in the table.

Our approach involves using heuristics to minimize costs for each of the two phases.

Approach for Phase One: To understand the complexity of optimizing the cost for this phase, let us assume that we know the cost for the operator $SF(A_j, \alpha)$. Our goal is to find the set of operations which has the least cost in coloring all columns of the table. This problem can be generalized and formulated as follows. For a set $S = \{S_1, \dots, S_n\}$, $S_1 \cup \dots \cup S_n = \{1, \dots, m\}$, where each set S_i has a cost function and corresponds to a chain set in the table M , we need to find the a subset of S who can cover $\{1, \dots, t\}$ with the least cost. This is a generalized *set-covering problem*, and is *NP-hard* [9].

Note, in our case, each row only needs at most one invocation of the SF operator, due to the *containing relation*. Therefore, we can enumerate the coloring schemes and find the one with the minimal cost in $O(j_1 \times \dots \times j_m) = O(t^m)$ time complexity. Here, m and t are the number of rows and columns respectively in table M , and j_i is the number of different support levels in the row i .

In practice, the size of the table is usually small, so the above enumeration can be done without a very high cost. However, the problem still is that precise cost functions are not available.

The heuristic approach we use is based on the observation that no repetitive computation due to the SF operator is involved in the phase one. So, we can solely focus on reducing the unnecessary computation. A natural heuristic for minimizing unnecessary computation is through the support level. For a single dataset, higher support level for the SF operator implies lower unnecessary computation. We use this in our implementation.

Input: table M after phase-one coloring

Algorithm 1

- Find datasets whose corresponding rows has black cells;
- For each row, find the lowest support level among black cells;
- On each row, we invoke the CF operator with the lowest support level. Across the rows, this operator is invoked in the decreasing order of support level used for the CF operator.

Algorithm 2

- Remove all the red cells from each chain set $S_{i,r}$;
- Find the non-empty chain set with the highest support and invoke the CF operator to color the set;
- Remove all new red cells from the chain set;
- Repeat the above steps until all cells are colored.

Figure 1. Algorithms for Phase Two

Approach for Phase Two: We can use either of the two greedy algorithms, *Algorithm 1* and *Algorithm 2*, which are listed in the Figure 1. The first algorithm tries to reduce the repetitive computation by invoking CF operator for each dataset at most once. Therefore, frequency of any itemset will

Input: table M without coloring

Algorithm 3

Build a collection of candidate sets by running the enumeration algorithm for $SF(A_j, \alpha)$ operator;
 For the candidate set S , let $SF(A_j, \alpha) \in S$
 and corresponding to the chain set $S_{j,r}$. If there exists another chain set $S_{j',r'}$ equivalent to $S_{j,r}$, transform $SF(A_j, \alpha) \in S$ into $GF(\{\langle A_j, \alpha \rangle, \langle A_{j'}, \alpha_{r'} \rangle\})$.
 Repeat the above step to see if any more set can be aggregated into a GF operation;
 Select a set from these transformed candidate sets based on some heuristic, e.g., the average size of the parameter set Y for the GF operation.

Figure 2. Using GF operator for Phase One

be counted at most two times for a dataset: one from the SF operator in the phase one and second from the CF operator in the phase two. However, much unnecessary computation is involved since CF operator always picks the lowest support level for each dataset. The second algorithm targets the unnecessary computation, since for each support level, CF operator will use the smallest possible set X to constraint the itemset generation. However, much repetitive computation can be generated, since an itemset can be computed several times for a dataset.

Let us consider the query Q . Combining phase one and phase two, the first algorithm gives the following query plan.

Phase1 : $SF(A, 0.1), SF(C, 0.1)$;
 Phase2 : $CF(A, 0.05, SF(C, 0.1)^I)$;
 $CF(B, 0.05, (SF(A, 0.1) \sqcup SF(C, 0.1))^I)$;
 $CF(D, 0.05, ((SF(A, 0.1) \sqcap SF(B, 0.1)) \sqcup SF(C, 0.1))^I)$;
 $CF(C, 0, (SF(A, 0.1) \sqcap SF(B, 0.1))^I)$;

The second algorithm gives the following query plan.

Phase1 : $SF(A, 0.1), SF(C, 0.1)$;
 Phase2 : $CF(B, 0.1, SF(A, 0.1)^I)$;
 $CF(D, 0.1, SF(C, 0.1)^I)$;
 $CF(A, 0.05, (SF(C, 0.1) \sqcap SF(D, 0.1))^I)$;
 $CF(B, 0.05, (SF(C, 0.1) \sqcap SF(D, 0.1))^I)$;
 $CF(D, 0.05, (SF(A, 0.1) \sqcap SF(B, 0.1))^I)$;
 $CF(C, 0, ((SF(A, 0.1) \sqcap SF(B, 0.1))^I)$;

We can see that both query plans can reduce the costs by aggressively utilizing the available information and the CF operator.

6.3.2 Using the Group Operator

The group mining operator GF can help remove some unnecessary computation due to SF operator. In the above example, suppose that $SF(A, 0.1) \sqcap SF(B, 0.1)$ and $SF(C, 0.1) \sqcap SF(D, 0.1)$ are generated in phase one. In this way, each column is also covered, and the unnecessary computation of set $SF^I(A, 0.1) - (SF(A, 0.1) \sqcap SF(B, 0.1))^I$ on dataset A is also saved.

	Query Conditions	CSET
Q_1	$A \geq \alpha_1 \wedge B \geq \alpha_1 \wedge C \geq \alpha_2 \wedge D \geq \alpha_2$	
Q_2	$(A \geq \alpha_1 \wedge B < \alpha_2) \vee (B \geq \alpha_1 \wedge A < \alpha_1)$	{A}
Q_3	$(A \geq \alpha_1 \wedge B \geq \alpha_1 \wedge C < \alpha_2 \wedge D < \alpha_2) \vee (C \geq \alpha_1 \wedge D \geq \alpha_1 \wedge A < \alpha_2 \wedge B < \alpha_2)$	{C}
Q_4	$(A \geq \alpha_1 \vee B \geq \alpha_1 \vee C \geq \alpha_1) \wedge D < \alpha_2$	{D}
Q_5	$A \geq \alpha_1 \wedge B \geq \alpha_1 \wedge C \geq \alpha_1 \wedge D < \alpha_2$	{D}
Q_6	$(A \geq \alpha_1 \wedge B < \alpha_2 \wedge C < \alpha_2 \wedge D < \alpha_2) \vee (B \geq \alpha_1 \wedge A < \alpha_2 \wedge C < \alpha_2 \wedge D < \alpha_2) \vee (C \geq \alpha_1 \wedge A < \alpha_2 \wedge B < \alpha_2 \wedge D < \alpha_2) \vee$	{D}

Table 7. Test Queries for Our Experiments

The use of GF operator only changes the phase one, i.e, our method for coloring at least cell in each column. Instead of finding $SF(A_j, \alpha)$ operations to cover each column, we now need to find GF operations to meet the same goal. *Algorithm 3*, described in Figure 2, uses the GF operator in a efficient way. It results in the following query plan for our example query:

Phase1 : $GF(\{\langle A, 0.1 \rangle, \langle B, 0.1 \rangle\})$;
 $GF(\{\langle C, 0.1 \rangle, \langle D, 0.1 \rangle\})$;
 Phase2 : $CF(A, 0.05, (SF(C, 0.1) \sqcap SF(D, 0.1))^I)$;
 $CF(B, 0.05, (SF(C, 0.1) \sqcap SF(D, 0.1))^I)$;
 $CF(D, 0.05, (SF(A, 0.1) \sqcap SF(B, 0.1))^I)$;
 $CF(C, 0, ((SF(A, 0.1) \sqcap SF(B, 0.1))^I)$;

7 Experimental Evaluation

This section reports a series of experiments we conducted to demonstrate the efficacy of the optimization and transformation techniques we have developed. Particularly, we were interested in the following questions:

1. What are the performance gains from the use of new mining operators, CF and GF , and what are the key factors impacting the level of gain.
2. Compared with the naive evaluation method, what performance gains are obtained from the of different optimizations, and new query plans generated using the three algorithms we have presented.

Initially, we briefly describe how the three new operators we introduced were implemented.

7.1 Implementation of Operators

The operators used in our query evaluation are the *frequent mining operator*, the *counting operator*, the *frequent itemset with constraints operator*, and the *group frequent itemset operator*. For our experimental study, Borgelt’s implementation of the well-known Apriori algorithm [5] is used as the frequent mining operator. The other three operators were derived from it as follows:

Counting operator $P(X, A_j)$: Initially, the set of itemsets X is organized as a prefix tree, where each node corresponds to an itemset. Then, a single pass on the dataset A_j is taken to project each transaction onto the prefix tree, using a depth-first traversal.

Frequent itemset mining operator with constraints: $CF(A_j, \alpha, X)$: Initially, the set of itemsets X is put into a hash table. The processing of CF is similar to the frequent itemset mining operator, with one exception in the candidate

Query	Naive	ORR	CF-1	CF-2	GF-1
$Q_1(60\%, 40\%)$	397		168		158
$Q_2(60\%, 40\%)$	626	352	158		
$Q_3(60\%, 40\%)$	914	619	236	386	277
$Q_1(50\%, 35\%)$	1024		279		265
$Q_2(50\%, 35\%)$	1381	687	265		
$Q_3(50\%, 35\%)$	2206	1558	394	484	471

Table 8. Performance (in seconds) on IPUMS datasets

Query	Naive	ORR	CF-1	GF-1
$Q_4(85\%, 55\%)$	1229	1085		
$Q_5(85\%, 55\%)$	1178	1032	301	218
$Q_6(85\%, 55\%)$	2502	1350	1304	
$Q_4(60\%, 40\%)$	1525	1361		
$Q_5(60\%, 40\%)$	1313	1152	491	216
$Q_6(60\%, 40\%)$	2634	1392	1470	

Table 9. Performance (in seconds) on DARPA datasets

generation stage. While placing an itemset in the candidate set, not only all its subsets need to be frequent, but the itemset needs to be in the hash table as well.

Group frequent itemset mining operator $GF(Y)$: The parameter $Y = \{ \langle A_1, \alpha_1 \rangle, \dots, \langle A_u, \alpha_u \rangle \}$, specifies the support level α_i for the dataset A_i . There are three differences between the implementation of this operator and the implementation of the common frequent mining operator. First, each node representing an itemset in the prefix tree has one count field for each dataset in Y . Second, the counts for each dataset are updated independently. Finally, in the candidate generation stage, an itemset is treated as a candidate set if all of its subsets are frequent in every dataset in Y .

7.2 Datasets

Our experiments were conducted using three groups of data, each of them comprising four different datasets.

IPUMS: The first group of datasets is derived from the *IPUMS* 1990-5% census micro-data, which provides information about individuals and households [1]. The four datasets each comprises 50,000 records, corresponding to New York, New Jersey, California, and Washington states, respectively. Every record in the datasets has 57 attributes. After discretizing the numerical attributes, the datasets have a total of 2,886 distinct items.

DARPA’s Intrusion Detection: The second group of datasets is derived from the first three weeks of *tcpdump* data from the DARPA data sets [26]. The three datasets include the data for three most frequently occurring intrusions, *Neptune*, *Smurf*, and *Satan*. The first two are Denial of Service attacks (DOS) and the last one is a type of Probe. Further, an additional dataset includes the data of the *normal* situation (i.e., without intrusion). Each transaction in the datasets has 40 attributes, corresponding to the fields in the TCP packets. After discretizing the numerical attributes, there are a total of

Query	Naive	ORR	CF-1	CF-2	GF-1
Q_1	3825		727		338
Q_2	7048	3384	1138		
Q_3	10369	7617	1344	1462	977
Q_4	2828	1395			
Q_5	2753	1324	693		283
Q_6	10105	7368	1815		

Table 10. Performance (in seconds) on QUEST datasets with query parameters $\alpha_1 = 0.3\%$ and $\alpha_2 = 0.1\%$

Query	Naive	ORR	CF-1	CF-2	GF-1
Q_1	5120		971		351
Q_2	9016	4379	1599		
Q_3	13285	9764	1743	1827	1042
Q_4	3823	2039			
Q_5	3662	1876	904		364
Q_6	13034	9394	2511		

Table 11. Performance (in seconds) on QUEST datasets with query parameters $\alpha_1 = 0.25\%$ and $\alpha_2 = 0.08\%$

343 distinct itemsets. The *neptune*, *smurf*, *satan*, and *normal* datasets contain 107,201, 280,790, 1,589, and 97,277 records, respectively.

IBM’s Quest: The third group of datasets represents the market basket scenario, and is derived from IBM Quest’s synthetic datasets [2]. The first two datasets, *dataset-1* and *dataset-2*, are generated from the *T20.I8.N2000* dataset by some perturbation. Here, the number of items per transactions is 20, the average size of large itemsets is 8, and the number of distinct items is 2000. For perturbation, we randomly change a group of items to other items with some probability. The other two datasets, *dataset-3* and *dataset-4*, are similarly generated from the *T20.I10.N2000* dataset. There are a total of 1943 distinct items in the four datasets, and each of them contains 1,000,000 transactions.

7.3 Test Queries

Our experiments use six different queries, which are listed in the Table 7. The first three queries, Q_1 , Q_2 , and Q_3 , are applicable on IPUMS datasets, and the New York, New Jersey, California, and Washington datasets are labeled as the datasets *A*, *B*, *C*, and *D*, respectively. The other three queries, Q_4 , Q_5 , and Q_6 , correspond to the queries in the motivating example on finding the signature itemsets for network intrusion, presented in Section 2. The *neptune*, *smurf*, *satan*, and *normal* datasets are labeled as the datasets *A*, *B*, *C*, and *D*, respectively. Further, in the Table 7, the *CSET* is specified. Finally, each query requires two different support levels, α_1 and α_2 . The evaluation using the IBM Quest dataset used all six queries.

In our experiments, up to five different query plans were implemented for each query. The exact number depended upon the applicability of specific optimization strategies on the given query. The five query plans are as follows:

1. Naive: using the naive evaluation method.
2. ORR: applying *Optimization RR* and *Transformation 1* to remove the negative predicate.
3. CF-1: applying the constraint frequent itemset mining operator *CF* and using the *Algorithm 1*.
4. CF-2: applying the constraint frequent itemset mining operator *CF* and using the *Algorithm 2*.
5. GF-1: applying the group frequent itemset mining operator *GF* and using the *Algorithm 3* (in Phase 1, and *Algorithm 1* in Phase 2).

7.4 Experimental Results

This subsection reports the results we obtained. All experiments are performed on a 933MHZ Pentium III machine with 512 MB main memory.

Table 8 presents the running time for the first three queries on IPUMS datasets. Table 9 shows the results from the other three queries, Q_4 , Q_5 , and Q_6 , on DARPA datasets. Also, all six queries were used with the QUEST synthetic datasets, and the results are presented in Tables 10 and 11. Each query is executed with two different pairs of support levels.

The queries Q_1 and Q_5 mainly show how the *CF* and *GF* operators can reduce the evaluation cost. The *CF* operator amounts to an average of more than 3 times speedup on both real and synthetic datasets. The speedups are higher with Query Q_1 than query Q_5 , since the *CF* operator is applied three times in Q_1 and only two times in Q_5 . Further, the *GF* operator performs better than *CF* operator for both the queries, and gains an average of 4 times the speedup on the real datasets, and up to 14 times speedup on the synthetic datasets.

The queries Q_2 , Q_3 , Q_5 , and Q_6 benefit from the *Optimization RR* and are able to use the *CF* operator. The ORR versions can achieve up to two times the speedup in these cases, and CF-1 always performs better than ORR. The query plan CF-1 can achieve an additional speedup of more than 5. Further, in all test cases, the versions CF-1 perform a little better than the version CF-2. This suggests that in the phase two, reducing the repetitive computation is more important. At last, the query Q_4 can be optimized by removing the negative predicate, but the *CF* and *GF* operators cannot be applied.

The results from the query Q_6 give rise to the following question: “*Why does the GF-1 query plan perform better than the CF-1 plan on QUEST datasets, and CF-1 performs better than GF-1 on IPUMS datasets*”. A related issue is that depending on the datasets and queries, the performance gains from the *CF* and *GF* operators can vary significantly. For example, the difference in speedup varies from 3 to 14 in our experiments. By further analyzing the detailed cost of each query, we believe that one of the key factors impacting the performance gains from both *CF* and *GF* operators is the ratio of the size of the intersection set with size of the set generated directly from the common frequent itemset mining operator. The less the ratio is, the more gain we can get from the *GF* operator by reducing the unnecessary computation

and lesser repetitive computation is introduced. For example, in the query Q_1 (50%, 35%) on IPUMS datasets, the size of intersection set is 19 times smaller than the total size of the four sets of frequent itemsets. However, in query Q_1 on QUEST synthetic datasets, the size of the intersection set is more than 1000 times smaller than the total size of the four sets of frequent itemsets.

To summarize, the new query plans CF-1 and GF-1 do result in improved performance, provided they are applicable on a given query. In our experiments, they show an improvement ranging from a factor of 2 to 15. Moreover, the size of intersection set is a significant factor impacting the performance gains from the use of *CF* and *GF* operators.

8 Related Work

Much research has been conducted to provide database support for mining operations. Han, Meo, Imielinski, and their colleagues have proposed extensions of the database query languages to support mining tasks [14, 17, 24]. Sarawagi and Agrawal [31] and Chaudhuri and his colleagues [8] have studied implementing Apriori association mining algorithm and decision tree construction, respectively, on a database system. ATLAS [37] applies user-defined functions (UDFs) to express data mining tasks. However, all of these efforts focus on mining a single dataset with relatively simple conditions.

A number of constraint frequent itemset mining algorithms have been developed to use additional conditions and prune the search space [6, 20, 22, 25, 28, 32]. However, these algorithms cannot efficiently answer our queries, since the conditions in our queries corresponds to a set of (in)frequent itemsets. These cannot be directly used to reduce the search space with their methods. We have developed a systematic approach for finding efficient query plans answering these queries.

Raedt and his colleagues have studied the generalized inductive query evaluation problem [21, 23, 30]. Although their queries target multiple datasets, they focus on the algorithmic aspects to apply version space tree and answer the queries with the generalized monotone and anti-monotone predicates. In comparison, we are interested in answering queries involving frequency predicates more efficiently. We have developed a table based approach to generate efficient query plans.

Our research is also different from the work on *Query flocks* [33]. While they target complex query conditions, they allow only a single predicate involving frequency, and on a single dataset. The work on multi-relational data mining [4, 12, 29, 36] has focused on designing efficient algorithms to mine a single dataset materialized as a multi-relation in a database system.

Finally, a number of researchers have developed techniques for mining the difference or *contrast sets* between the datasets [3, 11, 27, 35]. Their goal is to develop efficient algorithms for finding such a difference, and they have primarily focused on analyzing two datasets at a time. In comparison, we have provided a general framework for allowing the users to compare and analyze the patterns in multiple datasets.

Moreover, because our techniques can be a part of a query optimization scheme, the users need not be aware of the new algorithms or techniques which can speedup their tasks.

9 Conclusions and Future Work

The work presented in this paper is driven by two basic observations. First, analyzing and comparing patterns across multiple datasets is critical for many applications of data mining. Second, it is desirable to provide support for such tasks as part of a database or a data warehouse, without requiring the users to be aware of specific algorithms that could optimize their queries.

We have presented a systematic approach for expressing and optimizing frequent itemset queries that involve complex conditions across multiple datasets. Specifically, we have proposed an SQL-based mechanism and have established an algebra for such queries. We have developed a number of new optimizations, new operators, transformations, and heuristic algorithms for finding query plans with reduced execution costs. Our experiments have demonstrated up to an order of magnitude performance gains on both real and synthetic datasets. Thus, we believe that our work has provided an important step towards building an integrated, powerful, and efficient KDDMS.

Our future work will concentrate on several issues that remain open. First, our techniques could be extended to support *multiple query optimization* problem, and independent queries could be optimized simultaneously or incrementally. Second, our techniques will need some modifications for dealing with *evolving* datasets, or where there is a temporal order between the datasets. Third, providing cost functions for mining operators is an open and important issue. Fourth, implementing new mining operators efficiently needs more work. Finally, research is needed to incorporate other conditions, such as those defined in constraint itemsets mining, and apply other mining operators, such as the *maximal* frequent itemset operator.

References

- [1] Integrated public use microdata series. <http://http://www.ipums.umn.edu/usa/index.html>.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of Int. conf. Very Large DataBases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.
- [3] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Min. Knowl. Discov.*, 5(3):213–246, 2001.
- [4] Hendrik Blockeel and Michèle Sebag. Scalability and efficiency in multi-relational data mining. *SIGKDD Explor. Newsl.*, 5(1):17–30, 2003.
- [5] Christian Borgelt. Apriori implementation. <http://fuzzy.cs.Uni-Magdeburg.de/borgelt/Software-Version 4.08>.
- [6] Cristian Bucila, Johannes Gehrke, Daniel Kifer, and Walker White. Dualminer: a dual-pruning algorithm for itemsets with constraints. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 42–51, 2002.
- [7] T. Calders and J. Wijsen. On monotone data mining languages. In *Proc. of International Workshop on Database Programming Languages (DBPL)*, pages 119–132, 2001.
- [8] Surajit Chaudhuri, Usama M. Fayyad, and Jeff Bernhardt. Scalable classification over sql databases. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Australia*, pages 470–479. IEEE Computer Society, 1999.
- [9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, 1990.
- [10] Luc De Raedt. A perspective on inductive databases. *SIGKDD Explor. Newsl.*, 4(2):69–77, 2002.
- [11] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: discovering trends and differences. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 43–52, 1999.
- [12] Sašo Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explor. Newsl.*, 5(1):1–16, 2003.
- [13] F. Giannotti, G. Manco, D. Pedreschi, and F. Turini. Experiences with a logic-based knowledge discovery support environment. In *In Proc. 1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 1999)*.
- [14] J. Han, Y. Fu, W. Wang, K. Koperski, and O. R. Zaiane. Dmql: A data mining query language for relational databases. In *In Proc. 1996 SIGMOD 96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD 96)*, pages 27–33, Montreal, Canada, Jun 1996.
- [15] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2000.
- [16] Jiawei Han, Laks V. S. Lakshmanan, and Raymond T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.
- [17] T. Imielinski and A. Virmani. Msql: a query language for database mining. In *Data Mining and Knowledge Discovery*, pages 3:393–408, 1999.
- [18] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 39(11):58–64, 1996.
- [19] T. Johnson, Laks V. S. Lakshmanan, and Raymond T. Ng. The 3w model and algebra for unified data mining. In *Proceedings of International Conference on Very Large DataBases (VLDB)*, 2002.
- [20] Daniel Kifer, Johannes Gehrke, Cristian Bucila, and Walker White. How to quickly find a witness. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 272–283, 2003.
- [21] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 136–143, 2001.
- [22] Laks V. S. Lakshmanan, Raymond Ng, Jiawei Han, and Alex Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 157–168, 1999.
- [23] Sau Dan Lee and Luc De Raedt. An algebra for inductive query evaluation. In *Proc. The Third IEEE International Conference on Data Mining (ICDM'03)*, pages 147–154, Melbourne, Florida, USA, November 2003.
- [24] R. Meo, G. Psaila, and S. Ceri. A new sql-like operator for mining association rules. In *In Proc. of International Conference on Very Large Data Bases (VLDB)*, pages 122–133, Bombay, India, 1996.
- [25] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 13–24, 1998.
- [26] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. Towards nic-based intrusion detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 723–728, 2003.
- [27] Matthew E. Orey, Srinivasan Parthasarathy, Chao Wang, Adriano Veloso, and Wagner Meira Jr. Mining frequent itemsets in distributed and dynamic databases. Technical report, Department of Computer Science and Engineering, OSU, 2003.
- [28] Jian Pei, Jiawei Han, and Laks V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering*, pages 433–442, 2001.
- [29] Chang-Shing Perng, Haixun Wang, Sheng Ma, and Joseph L. Hellerstein. Discovery in multi-attribute data with user-defined constraints. *SIGKDD Explor. Newsl.*, 4(1):56–64, 2002.
- [30] Luc De Raedt, Manfred Jaeger, Sau Dan Lee, and Heikki Mannila. A theory of inductive query answering (extended abstract). In *Proc. The 2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 123–130, Maebashi, Japan, December 2002.
- [31] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998.
- [32] Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining, KDD*, pages 67–73, 1997.
- [33] Dick Tsur, Jeffrey D. Ullman, Serge Abiteboul, Chris Clifton, Rajeev Motwani, Svetlozar Nestorov, and Arnon Rosenthal. Query flocks: a generalization of association-rule mining. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 1–12, 1998.
- [34] J.D. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2002.
- [35] Geoffrey I. Webb, Shane Butler, and Douglas Newlands. On detecting differences between groups. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 256–265, 2003.
- [36] X. Yin, J. Han, J. Yang, and P. S. Yu. Crossmine: Efficient classification across multiple database relations. In *Proc. 2004 Int. Conf. on Data Engineering (ICDE'04)*, Boston, MA, March 2004.
- [37] Y.N. Law, C.R. Luo, H. Wang, and C. Zaniol. Atlas: a turing complete extension of sql for data mining applications and streams. In *Posters of the 2003 ACM SIGMOD international conference on Management of data*, 2003.