Preference-based Frequent Pattern Mining

Moonjung Cho *University at Buffalo*, mcho@cse.buffalo.edu

Jian Pei *Simon Fraser University*, jpei@cs.sfu.ca

Haixun Wang IBM *T.J Watson Research Center*,hanxun@us.ibm.com

Wei Wang *Fudan University*, weiwang1@fudan.edu.cn

ABSTRACT

Frequent pattern mining is an important data mining problem with broad applications. Although there are many in-depth studies on efficient frequent pattern mining algorithms and constraint pushing techniques, the effectiveness of frequent pattern mining remains a serious concern: *it is non-trivial and often tricky to specify appropriate support thresholds and proper constraints*.

In this paper, we propose a novel theme of *preference-based frequent pattern mining*. A user can simply specify a preference instead of setting detailed parameters in constraints. We identify the problem of preference-based frequent pattern mining and formulate the preferences for mining. We develop an efficient framework to mine frequent patterns with preferences. Interestingly, many preferences can be pushed deep into the mining by properly employing the existing efficient frequent pattern mining techniques. We conduct an extensive performance study to examine our method. The results indicate that preference-based frequent pattern mining is effective and efficient. Furthermore, we extend our discussion from pattern-based frequent pattern mining to preference-based data mining in principle and draw a general framework.

Keywords: Data mining, Frequent-pattern mining

1. INTRODUCTION

Frequent pattern mining (Agrawal et al., 1993) is an important data mining task with broad applications. In the last decade, there are many in-depth studies on frequent pattern mining, which can be grouped into two categories. The first category of studies (e.g., (Agrawal & Srikant, 1994; Han et al., 2000; Zaki et al., 1997a; Agarwal et al., 2001)) focus on developing efficient algorithms for frequent pattern mining. With the exciting progress, mining frequent patterns from large databases becomes highly feasible, though the development of faster algorithms is still demanding to catch up with the speed of data accumulation. On the other hand, it has been well recognized that the *effectiveness* of frequent pattern mining is a critical concern (Zheng et al., 2001). In many cases, frequent pattern mining may return such a huge number of frequent patterns that a user cannot handle. To tackle this problem, the constraint-based frequent pattern mining framework is proposed (Ng et al., 1998). Various constraints can be raised and only the frequent patterns satisfying the constraints should be mined. Efficient algorithms have been developed to push different kinds of constraints deep into the mining (e.g., (Ng et al., 1998;Lakshmanan et al., 1999; Pei & Han, 2000; Pei et al., 2001; Kifer et al., 2003)).

With the current frequent pattern mining algorithms and constraint pushing techniques, is frequent pattern mining effective and efficacious enough?

Example 1 (Motivating example 1) Suppose a manager in a large supermarket wants to find frequent patterns containing expensive items from customer transactions. In the constraint-based mining framework, to make the mining more selective, the manager may specify some constraints, such as *each pattern should contain an item with price more than* 100 *dollars and the total amount of each pattern should be at least* 500 *dollars*.

The quality of the above mining highly depends on whether the user can specify some proper constraints. However, the constraint specification is often challenging. For example, without a real test, it is hard to quantify “expensive item(s)" in a constraint. In practice, a user often has to adopt a make-do-and-mend approach. Finding the appropriate values for parameters in constraints by running the mining algorithms again and again is usually time consuming. ■

Example 2 (Motivating example 2) Consider mining frequent patterns for classification. Several recent studies (e.g., (Liu et al., 1998; Wang et al., 2000; Dong & Li, 1999; Li et l.,2001)) have shown that classification based on frequent patterns, such as associative classification and classification by emerging patterns, can achieve high accuracy and good understand-ability.

Intuitively, given a training data set where the records are grouped into two classes, positive samples C_+ and negative samples C_-, we want to find patterns frequent in one class and infrequent in the other. Moreover, the longer a frequent pattern (i.e., the more features a pattern covers), the better the predictability of a pattern. Thus, a user may specify a constraint on the length of the patterns. Again, quantifying the constraints is tricky. Without many real tests and fine tuning, there is no guarantee on the quality of the mining. ■

As shown in the above examples, even with the progress on efficient frequent pattern mining algorithms and constraint pushing techniques, the effectiveness of frequent pattern mining still remains a serious concern. *The major bottleneck is that a user has to specify the appropriate constraints, which is often beyond the user's knowledge about the data*.

In this study, we propose *preference-based frequent pattern mining,* a novel theme of frequent pattern mining. Instead of specifying solid constraints, a user can simply specify *preferences*. In Example 1, the manager in the supermarket can write a preference "*mine patterns from the transaction database, the more frequent a pattern and the more expensive the total amount of items in a pattern, the better the result. I definitely do not want to see any pattern whose frequency is less than* 0.1%." In Example 2, to build a classifier using frequent patterns, we may write a preference "*mine patterns from C_+ and C_-, the larger difference between the frequencies of a pattern in the two classes and the longer the pattern, the better the pattern is preferred. But I am not interested in any patterns that are more frequent in C_- than in C_+, or whose frequency is lower than* 0.1%.*"* Clearly, compared to constraints, preferences are much easier to write and capture the users' requirements more accurately.

In this paper, we study the problem of preference-based frequent pattern mining and make the following contributions. First, we identify the problem of preference-based frequent pattern mining, and formulate the preferences for mining. Second, we develop an efficient framework to mine frequent patterns with preferences. Interestingly, many preferences can be pushed deep into the mining by properly employing the existing efficient frequent pattern mining techniques. We conduct an extensive performance study to examine our method. The results indicate that preference-based frequent pattern mining is effective and efficient. Last, we extend our discussion from preference-based frequent pattern mining to preference-based data mining in principle and draw a general framework.

The rest of the paper is organized as follows. The problem of preference-based frequent pattern mining is described in Section 2. In Section 3, efficient mining algorithms are developed. An extensive performance study is reported in Section 4. We review related work and discuss a general framework of preference-based data mining in Section 5. Section 6 concludes the paper.

2. PREFERENCE-BASED FREQUENT PATTERN MINING

Let $I = \{i_1, \ldots, i_n\}$ be the set of *items*. An *itemset* (or *pattern*) is a subset of I. For the sake of brevity, we often write an itemset as a string of items and omit the parentheses. For example, itemset $\{a, b, c, d\}$ is written as *abcd*. The number of items in an itemset is called its *length*. That is, $len(X) = \|X\|$.

A *transaction* $T = (tid, X)$ is a tuple such that *tid* is a transaction identity and X is an itemset. A transaction database *TDB* is a set of transactions. A transaction $T = (tid, Y)$ is said *contain* itemset X if $X \subseteq Y$. Given a transaction database TDB, the support of itemset X, denoted as sup(X), is the number of transactions in TDB containing X, i.e., $sup(X) = \|\{T = (tid,Y) \mid (T \in TDB) \wedge (X \subseteq Y)\}\|$.

Given a transaction database TDB and a *minimum support threshold min sup*, an itemset X is said a *frequent pattern* if $sup(X) \geq min\_sup$. The *problem of frequent pattern mining* is to find the complete set of frequent patterns from the database.

In general, we define preferences as follows.

Definition 1 (Preference) A *preference order* $\succ_P$ is a partial order over 2^I, the set of all possible itemsets. For itemsets X and Y, X is said *(strictly) preferable to* Y if $X \succ_P Y$. ■

Problem definition. An itemset X is called a *preference pattern* (with respect to preference P), if there exists no any other itemset Y such that $Y \succ_P X$. Given a transaction database *TDB*, a preference P and a support threshold, the *problem of preference-based frequent pattern mining* is to find the complete set of preference patterns with respect to P that are frequent. ■

Now, let us consider how to write a preference. A simple preference such as a preference based on support or length of the itemsets can be written using an auxiliary function $f : 2^I \rightarrow R$ to define the preference order. For example, to prefer more frequent patterns to less frequent ones, we can have $X \succeq_{sup} Y$ if $sup(X) \geq sup(Y)$. As another example, to prefer longer patterns to shorter ones, we can have $X \succeq_{length} Y$ if $\|X\| \geq \|Y\|$.

It becomes tricky when a user wants to integrate more than one preference. For example, when a user prefers *either more frequent patterns or longer patterns*, can we just simply write the preference as "X $\succ=$ Y if (sup(X) $\geq$ sup(Y)) $\vee$ (len(X) $\geq$ len(Y))"? Unfortunately, relation $\succ=$ is not loyal to the user's real intension. In fact, $\succ=$ defined so is even not a partial order. For example, suppose *sup(abc) > sup(abcd)*. We have both *abc* $\succ$ *abcd* (due to the support inequality) and *abcd* $\succ$ *abc* (due to the length inequality).

A closer look at the user's preference indicates that it should be read as follows. *For any two patterns X and Y, if sup*(X) $\geq$ *sup*(Y), $||X|| \geq ||Y||$, *and at least one equality does not hold, then X is more preferable*.

Based on the above analysis, we define an *integration* operation among preference relations as follows.

Definition 2 (Integration of preferences) Let $\succ=_1, \ldots, \succ=_k$ be k preference orders. We define the *integration* of $\succ=_1, \ldots, \succ=_k$, denoted by $\succ= = (\succ=_1 \otimes \ldots \otimes \succ=_k)$ as follows. For itemsets X and Y , X $\succ=$ Y , if for any i ($1 \leq i \leq k$), X $\succ=_i$ Y. X $\succ$ Y if X $\succ=$ Y and there exists at least one i_0 ($1 \leq i_0 \leq k$) such that $X \succ_{io} Y$.

For example, the preference "*either more frequent patterns or longer patterns are preferred*" can be expressed as $\succ= = (\succ=_{sup} \otimes \succ=_{length})$.

Theorem 1 (Integration of preferences) *The integration of multiple preference relations is a strict partial order.*

Proof. We prove that, given partial orders $\succ=_1$ and $\succ=_2$, $(\succ=_1 \otimes \succ=_2)$ is a partial order. The cases of more than two partial orders can be proved by a simple induction on the number of integration operators, which is finite.

$\succ=$ is trivially irreflexive. Suppose $X \succ= Y$ and $Y \succ= X$. Then, for each $\succ=_i$, we have $X \succ=_i Y$ and $Y \succ=_i X$. Since $\succ=_i$'s are partial orders, $X = Y$. That is, $\succ=$ is antisymmetric. Suppose $X \succ= Y$ and $Y \succ= Z$. For each $\succ=_i$, $X \succ=_i Y$ and $Y \succ=_i Z$. Thus, $X \succ=_i Z$. That means $X \succ= Z$. The transitivity holds. ■

3. ALGORITHMS

In this section, we develop the algorithms for preference-based frequent pattern mining. We model the preference-based frequent pattern mining as a problem of search in a preference graph. Using this general framework, various optimizations can be incorporated to achieve efficient mining methods.

One important guideline in our design is to reuse the existing efficient mining techniques as much as possible. The breadth-first and depth-first search methods for frequent pattern mining have been studied extensively and substantially. Interesting, we show that many preferences can be pushed deep into the mining by properly employing the existing efficient frequent pattern mining techniques. Their effectiveness and efficiency have been justified and verified concretely in the previous work.

3.1 Preference graph

We begin with the definition of preference graph, which records the preference relation between itemsets.

Definition 3 (Preference graph) For a preference order $\succ=$, the *preference graph G* = (2^I ,*E*) is the transitive closure of $\succ$ on 2^I , where *I* is the set of items in the domain. An edge $Y \rightarrow X \in E$ if $X \succ Y$.[1] ■

Example 3 (Preference graph) Consider transaction database *TDB* in Figure 1 and preference $\succ$ "at each length level, the more frequent or more expensive the pattern, the more preferable".

Transaction database TDB

Tid	Itemset
10	abcd
20	acde
30	bdef
40	ace

Prices of items

Item	Price
a	10
b	20
c	30
d	40
e	50
f	60

Figure 1: A Transaction database TDB.

The preference graph *PG* is shown in Figure 2. If the support threshold is 1, then four patterns should be returned, *e, ac, de and ace*. They are underlined in the figure. ■

It is easy to see that a preference graph has the following property.

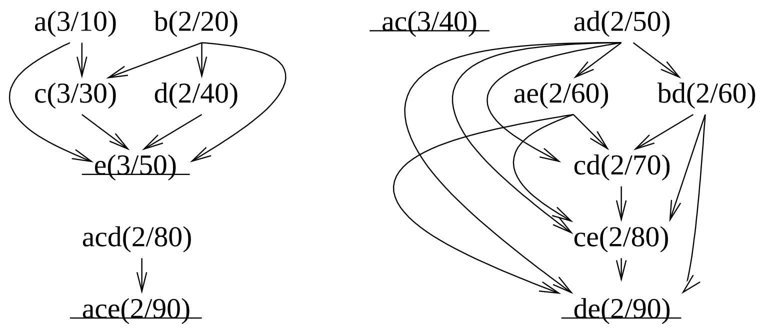

Figure 2: The preference graph *PG*. The numbers in the parentheses are (*sup/sum_price*).

Lemma 1 *A preference graph is an acyclic directed graph. An itemset X is a preference pattern if and only if X is a sink, i.e., the outdegree of X in the preference graph is 0.*

Proof. If X is a sink, there exists no any other itemset Y such that Y $\succ$ X. Thus, X is a preference pattern.

Suppose X is a preference pattern. If there is an edge Y$\rightarrow$X, then Y is preferable to X according to the definition of preference graph. That leads to a contradiction to the assumption that X is a preference pattern. ■

With the preference graph, the problem of preference-based frequent pattern mining is reduced to find the complete set of preferred patterns in the preference graph.

When mining frequent patterns with constraints, if the parameters of the constraints are inappropriate, we may get no pattern from the mining. *Is it possible that the similar situation happens in preference-based frequent pattern mining?*

Lemma 2 The search of a non-empty preference graph returns at least one preference pattern.

Proof. Every non-empty acyclic directed graph has at least one vertex of outdegree 0. Thus, if and only if the set of vertices in the preference graph is not empty, the preference graph must contain some preference pattern.

Lemma 2 indicates a nice property of frequent pattern mining with preference: applying a preference to the mining guarantees finding some answer. It also shows a major difference between preference-based and constraint-based mining.

3.2 Search in a Preference Graph

As shown in Example 3, a preference graph may or may not be connected, or even weakly connected[2]. An intuitive and basic method to search a preference graph systematically is as follows. We start from the vertices of indegree 0, and traverse every edge and thus every vertex. Now, let us justify such a strategy is feasible.

In a preference graph, an *undirected connected component* is a maximal subgraph such that for every pair of vertices *u*, *v* in the subgraph, there is an undirected path from *u* to *v*. Please note that an undirected connected component is weaker than a weakly connected component.[3]

Theorem 2 (Searchability) *In a preference graph, (1) each undirected connected component has at least one vertex of indegree 0; and (2) for each vertex u of indegree $k > 0$, there exists a vertex v of indegree 0 in the same undirected connected component such that there is a directed path from v to u.*

Proof. Since a preference graph is a directed acyclic graph, each undirected connected component is also a directed acyclic graph. Thus, we have the first half of the theorem immediately.

To show the second half of the theorem, we consider all vertices $P(u) = \{w | w \rightarrow u$ is an edge$\}$. If every vertex in $P(u)$ is of indegree $l > 0$, then there must be a directed cycle, which leads to a contradiction. Thus there must be a vertex $w_0 \in P(u)$ of indgree 0. Since $w_0 \rightarrow u$ is an edge, *u* and w_0 are in the same undirected connected component. ■

Based on Theorem 2, we can start from the vertices with indgree 0 and traverse the edges and vertices. The theorem says that such a search is complete.

It would be interesting to ask the following question. *Given a preference* $P = P_1 \otimes \cdots \otimes P_k$, *how many undirected connected components are there in the preference graph?*

Example 4 Let us consider the following preference on transaction database *TDB* in Figure 1: "*support is at least* 2, *the larger the better*, *the length is at least* 2, *and the larger the sum of price, the better*". The preference graph $PG_{Q'}$ is shown in Figure 3. As can be seen, in general the preference graph may contain more than one undirected connected component. ■

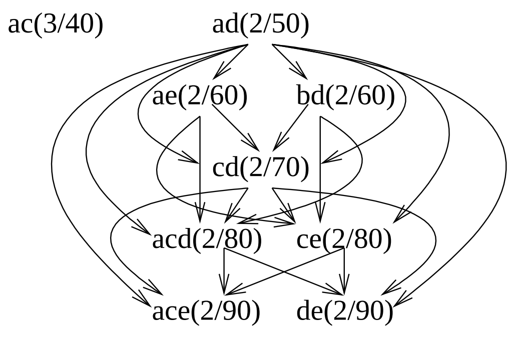

Figure 3: The preference graph $PG_{Q'}$. The numbers in the parentheses are (*sup/sum_price*).

3.3 Monotonic Preference Functions

There often exist a huge number of possible combinations of items in a large transaction database. All possible itemsets form a lattice according to the containment relation. For example, the itemset lattice of the transaction database *TDB* in Figure 1 is shown in Figure 4.

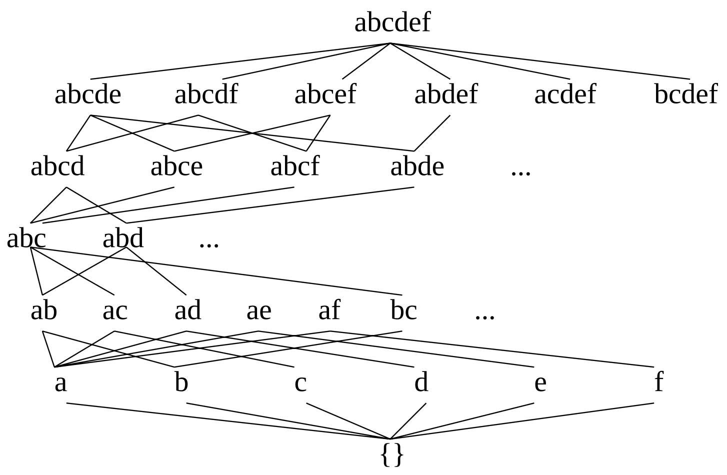

Figure 4: The itemset lattice.

Almost every frequent pattern mining method searches the itemset lattice from the bottom (i.e., $\emptyset$). One of the essential differences among various methods is the search strategies, which vary from breadth-first search to depth-first search, as well as their combinations (hybrid methods).

Now, the problem becomes, *"Given a preference, can we search the itemset lattice efficiently by pruning the unpromising branches using the preference?"*

Let us first consider the simplest case, where the preference is in the form of maximizing/minimizing a *preference function f(X)*.

Definition 4 (Monotonic preference function) A preference function $f(X)$ is called *monotonic increasing*, if for any itemsets X, Y such that $X \subset Y$, we have $f(X) \leq f(Y)$. The function is called *strictly monotonic increasing* if the equality always fails.

Similarly, a preference function $f(X)$ is called *monotonic decreasing*, if for any itemsets X, Y such that $X \subset Y$, we have $f(X) \geq f(Y)$. The function is called *strictly monotonic increasing* if the equality always fails.

A preference function is called *monotonic* if it is either monotonic increasing or monotonic decreasing. ■

The monotonicity of some commonly used functions is listed in Figure 5. The correctness of the table can be verified according to Definition 4 and the definitions of the functions.

Based on the monotonicity of the preference functions, we can predict whether a superset of an itemset can be a preference pattern. The idea is elaborated in the following example.

Example 5 (Pruning using monotonic functions) Suppose that we want to mine patterns with preference "*sup*(*X*) MAX". Since *sup*(*X*) is a monotonic decreasing function, for every superset $Y \supseteq X$, we have $sup(Y) \leq sup(X)$. If itemset *X* is not a preference pattern, then every superset of *X* cannot be a preference pattern, either. In other words, we even do not need to search any superset of *X* provided that *X* is not a preference function.

As another example, let us consider mining with preference "*sum*(*X*) MAX", where every item has a positive value. According to Figure 5, *sum*(*X*) is a monotonic increasing function. That is, for any itemsets *X* and *Y* such that $X \subseteq Y$, $sum(X) \leq sum(Y)$. If *Y* is not a preference pattern, then *X* cannot be a preference pattern, either.

Last, let us consider an example of integration of preferences. Consider preference "(*sum*(*X*) MAX) ⊗ (*len*(*X*) MAX)". It can be easily verified that, for itemsets *X* and *Y* such that $X \subseteq Y$, if *Y* is not a preference pattern, then *X* cannot be a preference pattern, either. ■

Function	Monotonicity
min(X)	monotonic decreasing
max(X)	monotonic increasing
sup(X)	monotonic decreasing
sum(X)	monotonic increasing if every item has a non-negative value monotonic decreasing if every item has a non-positive value otherwise, non-monotonic
avg(X)	non-monotonic
len(X)	strictly monotonic increasing

-f(X)	(strictly) monotonic increasing if *f(X)* is (strictly) monotonic decreasing (strictly) monotonic decreasing if *f(X)* is (strictly) monotonic increasing
f(X) + g(X)	(strictly) monotonic increasing/decreasing if both *f(X)* and *g(X)* are (strictly) monotonic increasing/decreasing
$\frac{1}{f(X)}$	(strictly) monotonic increasing/decreasing if *f(X)* is (strictly) monotonic decreasing/increasing
f(X) · g(X)	(strictly) monotonic increasing/decreasing if both *f(X)* and *g(X)* are positive and (strictly) increasing/decreasing

Figure 5: The monotonicity of some commonly used functions

In general, we have the following result.

Definition 5 (Monotonicity of preferences) A preference *P* is called *monotonic* if for any pattern *X*, *X* is not a preference pattern implies that every sub-pattern of *X* is not a preference pattern, either. A preference *P′* is called *anti-monotonic* if for any pattern *X*, *X* is not a preference pattern implies that every super-pattern of *X* is not a preference pattern, either. ■

Theorem 3 (Monotonic/anti-monotonic preferences) *Let f, f′ be monotonic increasing functions, and g, g′ be monotonic decreasing functions.*

1. Preferences $f(X)$ MIN, $g(X)$ MAX, ($f(X)$ MIN) ⊗ ($f'(X)$ MIN), ($g(X)$ MAX) ⊗ ($g'(X)$ MAX), *and* ($f(X)$ MIN) ⊗ ($g(X)$ MAX) *are anti-monotonic.*

2. Preference $f(X)$ MAX, $g(X)$ MIN, ($f(X)$ MAX) ⊗ ($f'(X)$ MAX), ($g(X)$ MIN) ⊗ ($g'(X)$ MIN), *and* ($f(X)$ MAX) ⊗ ($g(X)$ MIN) *are monotonic.*

Proof. The monotonicity of the preferences can be easily verified by the definitions. Limited by space, we omit the details here. ■

Now, we are ready to explore how to push preferences into frequent pattern mining. Most of the frequent pattern mining methods can be divided into two categories: breadth-first search approaches, such as Apriori (Agrawal & Srikant, 1994) and its enhancements, and depth-first search approaches, such as *FP-growth* (Han et al., 2000) and *TreeProjection* (Agarwal et al., 2001). In the next two subsections, We will study preference pushing in breadth-first search approaches and in depth-first search approaches, respectively.

3.4 Pushing Preferences into Breadth-first Frequent Pattern Mining

Apriori (Agrawal & Srikant, 1994) is a typical breadth-first search method of frequent pattern mining. It mines the frequent patterns by multiple scans of the transaction database.

In the first scan, it counts the support for every item. Then, infrequent items are discarded. Each pair of frequent items generates a length-2 *candidate itemset*. In the second scan, the support of every length-2 candidate itemset is counted. Those candidates failing the support threshold are discarded. A length-3 candidate itemset X is generated if every length-2 subset of X is frequent. Such a candidate-generation-and-test process continues. In general, in the k-th scan, length-k candidates are counted. Infrequent candidates are discarded. The frequent length-k candidates are used to generate length-(k +1) candidates. A length-(k +1) itemset X is a candidate if every length-k subset of X is frequent. The iteration proceeds until there is no candidate is frequent after one scan or there is no longer candidate can be generated from the current set of frequent patterns.

Many enhancements of Apriori work in the above framework, with various improvements on candidate generation or counting. Now, let us consider how to push a preference into the above breadth-first frequent pattern mining process.

Monotonic and anti-monotonic preferences can be pushed deep into the mining. If a preference is anti-monotonic, once an itemset X is not a preference pattern, any superset of X should not be generated as a candidate. By doing so, the number of candidates may be reduced.

A monotonic preference cannot be pushed into a breadth-first frequent pattern mining method directly. We need some auxiliary mechanisms. We illustrated the idea in the following example.

Example 6 (Pushing preference into breadth-first frequent pattern mining) Let us consider mining with preference $sum(X)$ MAX with support threshold 2 on the transaction database shown in Figure 1.

At the beginning, we generate 6 length-1 candidates, i.e., *a*, *b*, *c*, *d*, *e*, *f*. In the first scan of the database, we count the support of the candidates. Besides, we sort all the items alphabetically, and also count the support of *speculating candidates abcdef*, *bcdef*, *cdef*, *def* and *ef*. The benefit of counting these itemsets is clear. For example, if *abcdef* is a frequent pattern, it must be the longest one and thus the preference pattern. If so, the mining is done. For other speculating patterns, such as *cdef*, if they are frequent, any subset of them cannot be preference pattern and thus can be pruned from the candidate generation.

By the second scan of the database, we found that *f* is infrequent and none of the speculating patterns is frequent. We generate length-2 candidates, i.e., *ab*, *ac*, ... Moreover, we generate the speculating candidates *abcde*, *bcde* and *cde*. This procedure can continues until all preference patterns are found. ■

The above speculate-and-test idea was firstly used by Bayardo in mining max-patterns (Bayardo, 1998). Here, we extend the method to incorporate preferences.

In general, let R be a global order of items. In every itemset, we list all the items in the order of R. After the k-th scan of the database, the set of frequent k-patterns (i.e., the frequent itemsets of length k) are found. The *speculating candidates* are generated as follows. For every frequent k itemset $X = x_1 \cdots x_k$, let $\{Y_1, \cdots, Y_m\}$ be the complete set of frequent k-itemsets having prefix $x_1 \cdot x_{k-1}$. Suppose $Y_i = x_1 \cdots x_{k-1} y_i$ $(1 \leq i \leq m)$. Then, itemset $x_1 \cdots x_k y_1 \cdots y_m$ is a speculating candidate.

Based on the above discussion, we have the algorithm in Figure 6 to push preferences into the breadth-first frequent pattern mining.

Algorithm 1 (Pushing preference into breadth-first search)

Input: a mining query with a preference, the preference is either monotonic or anti-monotonic;

Output: the complete set of preference patterns;

Method:

```
conduct breadth-first search, in each round, do
    IF the preference is anti-monotonic
    THEN every superset of a non-preference pattern should not be generated as a
        candidate;
```

```
IF the preference is monotonic
THEN
    generate and test speculating candidates;
        for any a speculating candidate which is a non-preference pattern,
        its subsets should not be generated as candidates
```

Figure 6: The algorithm pushing preferences into breadth-first search

Pushing Preferences into Depth-first Frequent Pattern Mining

A typical depth-first frequent pattern mining method (e.g., FP-growth (Han et al., 2000) and TreeProjection (Agarwal et al., 2001)) works as follows.

First, the transaction database is scanned and all frequent items are identified. The frequent items are sorted in an order $R = x_1x_2 \cdot\cdot\cdot x_n$. The set of frequent patterns can be divided into n subsets: the ones containing x_1, the ones containing x_2 but not x_1, the ones containing x_3 but not x_1 or x_2, · · · , the pattern x_n.

To find patterns containing x_1, the x_1-projected database is formed by collecting the transactions containing x_1. Frequent items, except for x_1 itself, within the x_1-projected database are identified. For each frequent item x_j ($x_j \neq x_1$) within x_1-projected database, x_1x_j is a length-2 frequent pattern. The subset of frequent patterns containing x_1 can be further partitioned according to the frequent items within the x1-projected database, and the search can be conducted recursively.

To find patterns containing x_2 but not x_1, the x_2-projected database is formed by collecting the transactions containing x_2 and ignore the occurrences of x_1 in them. The x_2-projected database can be mined recursively.

Similarly, the remaining subsets of frequent patterns can be mined by forming appropriate projected databases and mining them recursively.

Conceptually, the depth-first frequent pattern mining conducts a depth-first search over the itemset lattice following a expanding tree based on the order R. For example, if the alphabetical order is taken in the depth-first mining of the itemset lattice in Figure 4, then the expanding tree is as shown in Figure 7.

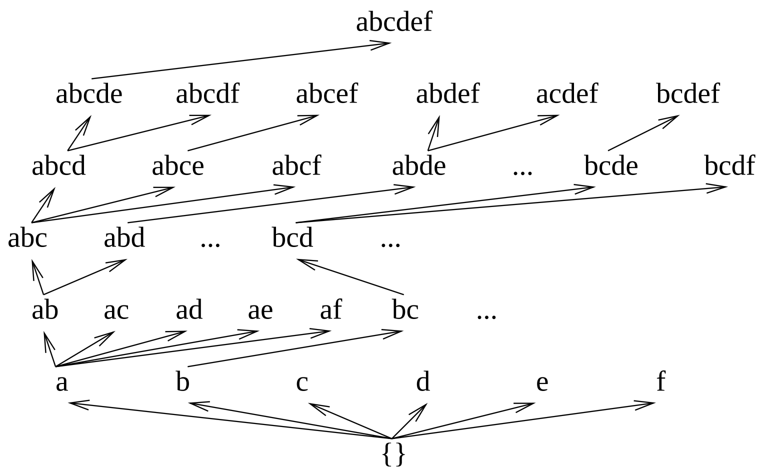

Figure 7: The expanding tree of the itemset lattice in Figure 4, where the alphabetical order is used.

How can we push a preference into the depth-fistrst frequent pattern mining? On the one hand, if a preference is anti-monotonic, then for any itemset X that is not a preference pattern, we can prune the whole sub-tree rooted at X (i.e., the itemsets having X as a subset). This follows the first case in Theorem 3.

On the other hand, if a preference is monotonic, then for any itemset X, we can compare X with the preference patterns already found. If there exists a preference pattern Y such that Y is preferable to X, then the subtree rooted at X can be pruned. Moreover, let Z be the frequent items in the X-projected database.

Based on the above discussion, we have the algorithm in Figure 8.

Algorithm 2 (Pushing preference into depth-first search)

Input and output: same as Algorithm 1;

Method:

```
conduct depth-first search, at each node, do
    IF the preference is anti-monotonic THEN
        stop recursion if the itemset at the current node is not a preference pattern;
    IF the preference is monotonic THEN
        compare the itemset $X$ in the current node to the preference patterns found so
```

```
                    far;
            IF $X$ is not a preference pattern THEN stop recursion
```

Figure 8: The algorithm to push preferences into depth-first search

Handling Non-monotonic Functions

Theorem 3 supports efficient pruning for preferences written in monotonic functions and in one of the forms listed in the theorem. Corresponding algorithms are developed. Now, the remaining problem is "*What can we do for a preference using a non-monotonic function?*"

There are two ways to handle the non-monotonic functions. On the one hand, as shown in (Pei et al., 2001), some non-monotonic functions, such as *avg*(), can be *"converte"* so that they partially have the monotonic property. Take *avg*() as an example. In general, neither $avg(X) \leq avg(Y)$ nor $avg(X) \geq avg(Y)$ always hold for any itemsets X, Y such that $X \subseteq Y$. However, if we sort items in value descending order and list items in any itemset in this order, then for any itemsets X, Y such that X is a prefix of Y, we have $avg(X) \geq avg(Y)$. Such functions are called *convertible monotonic functions* (Pei et al., 2001).

It is hard to push a preference with convertible monotonic functions into a breadth-first frequent pattern mining method, since the order of items is hard to incorporate into the mining. Fortunately, preferences involving convertible functions can be pushed deep into the depth-first frequent pattern mining methods. For example, to push a preference *avg*(*X*) MAX, we can use the value descending order and thus the function becomes convertible monotonic decreasing with respect to the order. We also use this order to generate the expanding tree. It can be easily verified that Algorithm 2 still holds. Limited by space, we omit the details here.

On the other hand, in the case that a function f is even not convertible monotonic, we may also find some monotonic or convertible monotonic function f' such that f' can bound f. We show the idea using the following example. Limited by space, we omit the formal results.

Example 7 (Using a monotonic function to bound a non-monotonic one) Consider aggregate function $tran\_avg(X) = \frac{\sum_{(tid,Y)\in TDB, X\subseteq Y} sum(Y)}{sup(Y)}$. In words, *tran_avg*(*X*) computes the average value of the transactions containing itemset *X*. Clearly, function *tran_avg* is neither monotonic nor convertible monotonic.

Suppose a user want to mine frequent patterns with support threshold 100 and prefer the patterns in transactions of high values. We can define an auxiliary function *top_100_tran_avg*(X) as the average value of the top-100 transactions containing itemset X. For any itemsets X, Y such that $X \subseteq Y$, every transaction containing itemset Y must also contain X. Thus, *top_100_tran_avg*(X) $\geq$ *top_100_tran_avg*(Y). That is, function *top_100_tran_avg*(X) is monotonic decreasing. Moreover, for any itemset X, *top_100_tran_avg*(X) $\geq$ *tran_avg*(X).

Therefore, instead of pushing the original preference, we can push a preference *top_100_tran_avg* (X) MAX. If the *top_100_tran_avg* value of the current pattern is smaller than the *tran_avg* value of a preference pattern, the *tran_avg* value of the current pattern must be also smaller than that of the preference pattern, and thus the current pattern can be pruned. When a preference pattern with respect to preference *top_100_tran_avg* (X) MAX is found, we can test whether it is a preference pattern with respect to *tran_avg*(X) MAX. ■

3.5 Handling Complex Integrations of Preferences

Using the integration operator, we can write complex preferences. Theorem 3 gives a guidance on how to push the integration of preferences in the form of $P_1 \otimes P_2$ into frequent pattern mining, where both P_1 and P_2 are preferences listed in one case in the theorem. However, in practice, we may have some complex situation. Let us consider the following example.

Example 8 (Pushing complex integration of preferences) Suppose we want to mine patterns with preference P = (*len*(X) MAX) $\otimes$ (max(X) MIN). The preference is neither monotonic nor anti-monotonic even though it is written in monotonic functions.

Preference P can be pushed deep into a depth-first search as follows. At each node of the search, we know the (frequent) items in the projected database. They are the items can be used to assemble longer patterns from the current pattern. Thus, we know the upper bound of *len*(X) and the lower bound of the value of items in X for any possible pattern X in the recursive search. Using this information, we can determine whether it is promising to find new preference patterns. If not, the current search branch can be pruned.

Similarly, the preference can also be pushed into a breadth-first search. The major idea is that we use the speculate candidate to prune if the upper bound of the *len*(X) and the lower bound of the minimum item value are subsumed by a preference pattern. Limited by space, we omit the details. ■

4. EMPIRICAL EVALUATION

In this section, we report an empirical evaluation on preference-based frequent pattern mining. We test our methods on both synthetic data sets and real data sets. Limited by space, we only report our results on some synthetic data sets and two real data sets. The results on other data sets are consistent.

We implement all the algorithms in C++. All the experiments are run on a PC with a Pentium 4 Processor at 2.0GHz and 512M main memory. The three data sets are as follows.

- *Synthetic data set D*: *T*20*I*10*D*10*k*. We use synthetic data sets generated by the well known IBM transactional data generator (Agrawal & Srikant, 1994). In particular, the data set *D*: *T*20*I*10*D*10*k* contains 10,000 transactions. The average length of the transactions is 20. It contains many long patterns.
- *Real data set Retail*. The data set is a subset of transactions collected from a retail company. It contains 100,000 transactions.
- *Real data set Gazelle*. It is a web store visit (clickstream) data set from Gazelle.com.[4] It contains 59,601 transactions, while there are up to 267 items per transaction. This data set is extensively used in as a benchmark of frequent pattern mining.

4.1 The effectiveness of preference-based mining

To illustrate the effectiveness of preference-based mining, we evaluate the preference preferring frequent, long patterns.

Data sets	D (*min_sup*=0.03%)		Retail (*min_sup*=0.02%)		Gazelle(*min_sup*=0.059%)	
length	# pref pat	sup (%)	# pref pat	sup (%)	# pref pat	sup (%)
1	1	8.64	1	5.132	1	5.987
2	1	2.68	1	2.235	1	2.020
3	2	2.05	1	1.895	1	0.700
4	11	1.90	1	1.729	1	0.344
5	1	1.77	1	1.637	1	0.176
6	1	1.67	1	1.572	1	0.116
7	1	1.46	1	1.524	2	0.094
8	1	1.37	2	1.476	1	0.089
9	2	1.07	1	1.432	3	0.081
10	1	1.03	1	0.904	1	0.081
11	1	0.97	1	0.868	1	0.076

12	1	0.62	1	0.562	1	0.069
13	1	0.30	1	0.353	1	0.064
14	1	0.17	1	0.337	40	0.060
15	3876	0.05	1	0.020	2	0.060
16	969	0.05			6	0.059
17	171	0.05				
18	19	0.05				
19	1	0.05				
20	2	0.04				
21	2	0.03				

Figure 9: The number of preference patterns of various lengths and their supports.

In Figure 9, we show the number of preference patterns of various lengths and their supports. To facilitate comparison, we report the *relative supports* of the patterns, i.e., $\frac{sup(X)}{\|TDB\|}\cdot 100\%$. To avoid the noise caused by patterns of very low support, we set the minimum support thresholds as indicated in the table.

From the table, we can observe some interesting features of the distribution of preference patterns.

1. In this particular example, at most length levels, we have only a very small number of preference patterns (often less than 3 in Figure 9.) In general, the number of preference patterns depends on the preference, the data set and related parameters. However, a common feature from all of our experiments is that *preference-based mining consistently returns a very small number of preference patterns comparing to the frequent pattern mining*. In other words, even though there are many patterns, mining with the preference may effectively presents only the patterns interesting to the users.
2. In this particular example, at some length levels, we may have many preference patterns, such as the preference patterns of length 15 to 18 in synthetic data set *D*, and the patterns of length 14 in real data set Galleze. We printed out the patterns and found that those are noisy patterns with very low supports. Similar phenomenon is also observed in some of other experiments on different preferences and data sets. When the support threshold is very low, there are many "*noisy*" preference patterns caused by coincident sharing among transactions. However, even though in such extreme cases, the number of preference patterns is still much smaller than the

number of frequent patterns at the same support level. We recommend that a small support threshold should be used as a constraint for preference-based mining to prune such noisy patterns.

3. Interestingly, in this particular mining query, those very long preference patterns are formed by small subsets of transactions sharing the patterns. In general, by observing the statistics of preference patterns, we may find some interesting clusters hidden in the data. Such clusters are often hard to find by conventional frequent pattern mining since their supports are too low.

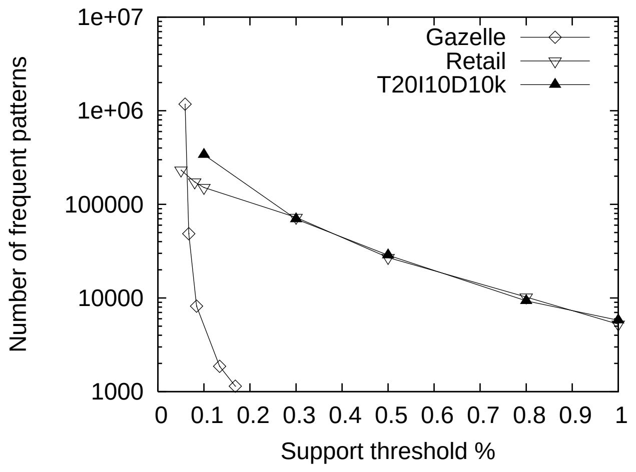

Figure 10: The number of frequent patterns with respect to support threshold.

Figure 10 plots the distribution of frequent patterns in the three data sets with respect to support threshold. In general, when the support threshold goes low, the number of frequent patterns increases dramatically. (Please note that the number of patterns is plotted in logarithmic scale.) Without preference-based mining, a user may have to go through hundreds of thousands or even millions patterns to find the preferable ones that are identified in Figure 9. Clearly, the effectiveness of the mining is the most important benefit of preference-based mining.

4.2 The efficiency of preference-based mining

The efficiency of pushing a preference into frequent pattern mining highly depends on the preference, the data set, and the related parameters (e.g., the available constraints). In Section 3, we developed some general techniques to push various preferences. As discussed before, the preference pushing techniques we developed in this paper have been used in the previous studies. In this paper, we generalize some of them and integrate them for pushing preferences into frequent pattern mining. The empirical evaluations reported in those previous studies (e.g., (Bayardo, 1998; Pei et al., 2001; Han et al., 2001)) fully indicate the efficiency of the techniques, respectively. Please note that, given a specific preference, it is highly possible that further pruning techniques with respect to the preference can be developed to improve the efficiency substantially. We do not discuss such specific techniques in this paper since the main task in this paper is to establish a general framework.

Limited by space, in this section, we only report results on an example case to illustrate the efficiency of pushing anti-monotonic preferences.

To test the efficiency of pushing anti-monotonic preferences, we use the mining query Q_{test}. Figure 11 shows the efficiency of the mining. The depth-first search implementation is based on FP-growth (Han et al., 2000), and the breadth-first search method is based on Apriori (Agrawal & Srikant, 1994). From the figure, we can observe that, in general, depth-first search is more efficient and can touch to lower support threshold than breadth-first search. We believe that the efficiency difference is from the inherent difference between depth-first search and breadth-first search, as extensively studied before (e.g., (Agarwal et al., 2001; Han et al., 2000; Zaki et al., 1997a)).

Data sets	Depth-first search runtime (seconds)	Breadth-first search runtime (seconds)
T20I10D10k	678.527 (*min_sup*=0.03%)	2893.843 (*min_sup*=0.2%)
Retail	404.629 (*min_sup*=0.02%)	2390.326 (*min_sup*=0.1%)
Gazelle	132.084 (*min_sup*=0.059%)	1245.232 (*min_sup*=0.1%)

Figure 11: The efficiency of pushing anti-monotonic preferences.

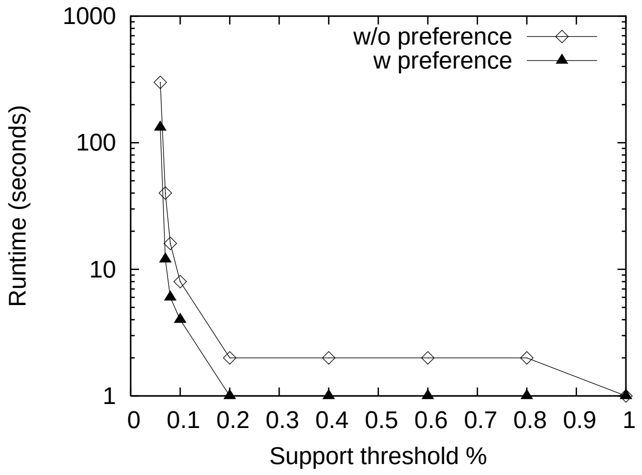

Figure 12: Comparison of the depth-first search runtime with and without preference Q_{test} on data set Gazelle.

To compare the mining efficiency with and without preferences, we plot Figure 12. The test is on depth-first search methods with and without preference Q_{test}. In order to show the difference of the runtime clearly, the runtime is plotted in logarithmic scale. From the figure, we can clearly see the pruning power of the techniques. The depth-first search with preference is always 2-3 times faster than the one without the preference.

We also test the scalability of the preference pushing techniques. As an example, we mine frequent patterns with preference Q_{test} on synthetic data set $T20I10D10 - 1000k$ of 10,000 to 1,000,000 transactions. The support threshold is set to 0.1%. The depth-first search is used. The result is shown in Figure 13. Roughly, the curve is linear. (It takes 7 seconds on the data set of 10, 000 transactions and 792 seconds on the data set of 1,000,000 transactions). We also test some other cases. They are all scalable with respect to the number of transactions in the data sets.

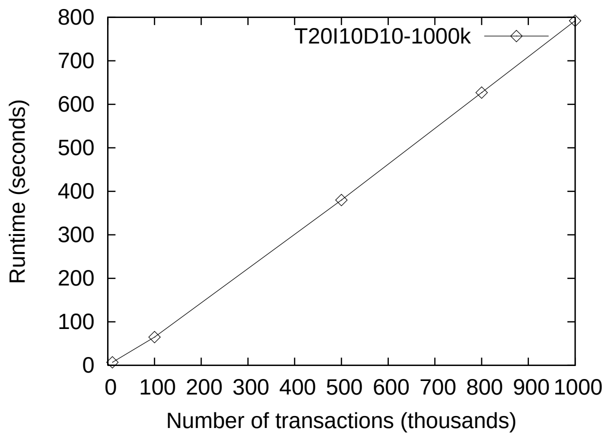

Figure 13: The scalability of preference pushing with respect to database size.

5. RELATED WORK AND DISCUSSION

In this section, we first briefly review related work. Then, we discuss a general framework of preference-based data mining.

5.1 Related Work

The research on preference-based frequent pattern mining is related to two categories of previous studies: preference queries and frequent pattern mining techniques.

The problem of preference queries is firstly addressed by Lacroix and Lavency in (Lacroix & Lavency, 1987). They formulate preferences as a series of conditions for selection.

The nonempty set of answer satisfying the longest sub-series of conditions should be returned. The compositions of preferences are not considered.

Several logical approaches to preferences are proposed in the context of deductive databases, including (Govindarajan et al., 1995; K. Govindarajan and B. Jayaraman and S. Mantha, 2001; Kostler et al., 1995). The formal languages of preference relations are developed in (Kiessling, 2002; Kiessling & Kostler, 2002; Chomicki, 2002). The idea of specifying preferece-based mining queries in this paper is similar to those in the above studies.

The problem of frequent pattern mining is firstly identified by Agrawal et al. as a key step in association rule mining (Agrawal et al., 1993). Many algorithms have been designed to mine frequent patterns efficiently, e.g., (Agrawal & Srikant, 1994; Han et al., 2000; Agarwal et al., 2001; Zaki et al., 1997b). As indicated by many studies (e.g., (Zheng et al., 2001)), frequent pattern mining often returns a huge number of patterns. To improve the effectiveness as well as the efficiency of frequent pattern mining, the problem of constraint-based frequent pattern mining is studied recently (e.g., (Ng et al., 1998; Lakshmanan et al., 1999; Pei et al., 2001)). Constraints are classified into various categories, e.g., monotonic, anti-monotonic, succinct, and convertible ones. Constraints in each category share some computational properties. Methods have been developed to push various constraints deep into the mining.

Some special cases of preference-based frequent pattern mining have been studied recently. For example, (Bayardo & Agrawal, 1999) proposes mining the most interesting rules. (Fu et al., 2000) studies mining the top-*k* frequent itemsets. In (Han et al., 2002), an efficient algorithm for mining the top-*k* frequent closed itemsets is proposed.

To the best of our knowledge, this paper is the first study systematically tackling the problem of preference-based frequent pattern mining. We address both the effectiveness and the efficiency of the preference-based frequent pattern mining. We provide simple and sufficient mechanisms to support users' specification of preferences and explore efficient methods to push preferences into the mining.

5.2 A General Framework of Preference-based Data Mining

The general idea of preference-based frequent pattern mining can be extended and generalized. Here, we propose a general framework of preference-based data mining.

In principle, a data mining query should contain the following three components.

1. *The specification of types of patterns*. For example, a user may say she/he wants to mine frequent patterns, association rules, classification rules/classifiers, or clusters. This component is obligatory.
2. *The specification of constraints*. This component is optional. If constraints present, only the patterns satisfying the constraints should be returned to the users.
3. *The specification of preferences*. This component is also optional. A preference specifies how to select the patterns most interesting to users from all the feasible answers satisfying all constraints.

To some extent, a preference can be regarded as a *soft* constraint. That is, the patterns best satisfy the preference are output.

Based on the above framework, two important directions for the future studies on preference-based data mining are as follows.

1. *Specification of preferences*. Concise and effective methods should be developed to facilitate users' specification of various preferences. In general, a user's preference may have multiple goals. Specifying complex preferences is far from trivial.
2. *Pushing preferences into mining*. Preference-based data mining may improve not only the effectiveness but also the efficiency of the mining. Proper preference pushing techniques should be developed to utilize the preference to prune unpromising searches.

6. CONCLUSIONS

In this paper, we study the problem of preference-based frequent pattern mining. A user can simply specify a preference instead of setting detailed parameters and constraints. We identify and formulate the problem and the preferences for mining. We propose a SQL-like query language construct for effective specification of preference-based frequent pattern mining queries. Using some typical examples, we show that preference-based frequent pattern mining queries can be used to describe users' mining requirements effectively. We develop an efficient method to mine frequent patterns with preferences. Instead of identifying various specific kinds of preferences and develop individual preference mining algorithms, our method is general and can handle many preferences. Interestingly, many preferences can be pushed deep into the mining by properly employing the existing efficient frequent pattern mining techniques. We conduct an extensive performance study to examine our method. The results indicate that preference-based frequent pattern mining is effective and efficient. Furthermore, we extend our discussion from pattern-based frequent pattern mining to preference-based data mining in principle and draw a general framework. Several interesting problems for future studies are discussed.

References

Agarwal, R. C., Aggarwal, C. C., & Prasad, V. V. V. (2001). A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing, 61*, 350-371.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)* (pp. 207-216). Washington, DC.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)* (pp. 487-499). Santiago, Chile.

Bayardo, R. J. (1998). Efficiently mining long patterns from databases. *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)* (pp. 85-93). Seattle, WA.

Bayardo, R. J., & Agrawal, R. (1999). Mining the most interesting rules. *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)* (pp. 145-154). San Diego, CA.

Chomicki (2002). Querying with intrinsic preferences. *Proc. 2002 Int. Conf. on Extending DataBase Technology (EDBT'02)* (pp. 34-51). Prague, Czech.

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)* (pp.43-52). San Diego, CA.

Fu, A. W.-C., Kwong, R. W.-W., & Tang, J. (2000). Mining n-most interesting itemsets. *Proc.200 Int. Symp. Methodologies for Intelligent Systems (ISMIS'00)* (pp. 59-67). Charlotte, NC.

Govindarajan, K., Jayaraman, B., & Mantha, S. (1995). Preference logic programming. *International Conference on Logic Programming* (pp. 731-745).

Han, J., Pei, J., Dong, G., & Wang, K. (2001). Efficient computation of iceberg cubes with complex measures. *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*(pp. 1-12). Santa Barbara, CA.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)* (pp. 1-12). Dallas, TX.

Han, J., Wang, J., Lu, Y., & Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*. Maebashi, Japan.

K. Govindarajan and B. Jayaraman and S. Mantha (2001). Preference Queries in Deductive Databases. *New Generation Computing*, 57-86.

Kiessling, W. (2002). Foundations of preferences in database systems. *Proc. 2002 Int. Conf. on Very Large Data Bases (VLDB'02)* (pp. 311-322). Hong Kong, China.

Kiessling, W., & Kostler, G. (2002). Preference sql - design, implementation, experience. *Proc.2002 Int. Conf. on Very Large Data Bases (VLDB'02)* (pp. 990-1001). Hong Kong, China.

Kifer, D., Gehrke, J., Bucila, C., & White, W. (2003). How to quickly find a witness. *Proc. 2003ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'03).*San Diego, California.

Kostler, G., Kiebling, W., Thone, H., & Guntzer, U. (1995). Fixpoint iteration with subsumption in deductive databases.

Lacroix, M., & Lavency, P. (1987). Preferences; putting more knowledge into queries. *VLDB'87,Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987,Brighton, England* (pp. 217-225). Morgan Kaufmann.

Lakshmanan, L. V. S., Ng, R., Han, J., & Pang, A. (1999). Optimization of constrained frequent set queries with 2-variable constraints. *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)* (pp. 157-168). Philadelphia, PA.

Li, W., Han, J., & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. *Proc. 2001 Int. Conf. Data Mining (ICDM'01)* (pp. 369-376). San Jose, CA.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. *Proc.1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)* (pp. 80-86). New York, NY.

Ng, R., Lakshmanan, L. V. S., Han, J., & Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)* (pp. 13-24). Seattle, WA.

Pei, J., & Han, J. (2000). Can we push more constraints into frequent pattern mining? *Proc.2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)* (pp. 350-354). Boston, MA.

Pei, J., Han, J., & Lakshmanan, L. V. S. (2001). Mining frequent itemsets with convertible constraints. *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)* (pp. 433-332). Heidelberg, Germany.

Wang, K., Zhou, S., & He, Y. (2000). Growing decision tree on support-less association rules. *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)* (pp. 265-269). Boston, MA.

Zaki, M. J., Parthasarathy, S., Ogihara, M., & Li, W. (1997a). New algorithms for fast discovery of association rules. *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)* (pp. 283-286). Newport Beach, CA.

Zaki, M. J., Parthasarathy, S., Ogihara, M., & Li, W. (1997b). Parallel algorithm for discovery of association rules. *Data Mining and Knowledge Discovery*, *1*, 343-374.

Zheng, Z., Kohavi, R., & Mason, L. (2001). Real world performance of association rule algorithms. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'01)* (pp. 401-406). San Francisco, California: ACM Press.

---

1 *Please note that we can draw preference graphs in the reserve way. That is, alternatively we can define the graph as edge $X \rightarrow Y \in E$ if $X \succ Y$. The drawing of preference graphs does not affect the mathematical meaning.*

2 *There are two distinct notions of connectivity in a directed graph. A directed graph is weakly connected if there is an undirected path between any pair of vertices, and strongly connected if there is a directed path between every pair of vertices.*

3 *A weakly connected component is a maximal subgraph of a directed graph such that for every pair of vertices* u, v *in the subgraph, there is an undirected path from u to v and a directed path from v to u*

4 *This data set is obtained from the KDD Cup 2001 website.*