RC 21990 (98808) 14 March 2001 **Computer Science**

IBM Research Report

FARM: A Framework for Exploring Mining Spaces with Multiple Attributes

Chang-Shing Perng, Haixun Wang, Sheng Ma, Joseph L. Hellerstein
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

FARM: A Framework for Exploring Mining Spaces with Multiple Attributes

Chang-Shing Perng Haixun Wang Sheng Ma Joseph L. Hellerstein
{perng,haixun,shengma,hellers}@us.ibm.com
IBM Thomas J. Watson Research Center
Hawthorne, NY 10532

Abstract

Mining for frequent itemsets typically involves a preprocessing step in which data with multiple attributes are grouped into transactions and items are defined based on attribute values. We have observed that such fixed attribute mining can severely constrain the patterns that are discovered. Herein, we introduce mining spaces, a new framework for mining multi-attribute data that not only discovers patterns but also discovers transaction and item definitions (with the exploitation of taxonomies and functional dependencies if they are available). We prove that special downward closure properties hold for mining spaces, a result that allows us to construct efficient algorithms for mining patterns without the constraints of fixed attribute mining. We apply our algorithms to synthetic data and to real world data collected from a production computer network. The results show that by exploiting the special kinds of downward closure in mining spaces, execution times for mining can be reduced by a factor of three to four.

1 Introduction

Mining for frequent itemsets has been studied extensively because of the potential for actionable insights. Typically, before mining is done, a preprocessing step uses data attributes to group records into transactions and to define the items used in mining. For example in supermarket data, the market basket attribute might be used to group data into transactions and the product-type attribute (with values such as diapers, beer) to specify items. We refer to this as **fixed attribute mining** in that mining does not change which attributes are used to determine transactions and items.

We have observed that fixing the attributes used to define transactions and items can severely constrain the patterns that are discovered. For example, by having items characterized in terms of product type, we may fail to discover relationships between baby items in general (e.g., diapers, formula, rattles) and adult beverages (e.g., beer and wine). And, by having transactions be market baskets, we may fail to note relationships between items purchased by the same family in a single day.

To go beyond the limits of fixed attribute mining, we introduce *FARM*, a new mining framework that

uses mining spaces to discover frequent patterns for transactions and items that are defined in terms of data attributes. Here a "transaction" is a general term for a group of records. Our framework does not require prespecified taxonomies, although our approach exploits such information if it is available. We prove that downward closure holds for a class of mining spaces. This results provides for the implementation of efficient mining algorithms. We apply our algorithms to event data collected from a production computer network and show that our approach is considerably faster than simply employing apriori-like algorithms on each choice of attributes for defining transactions and items.

To better motivate the problem we address, consider the domain of event management for complex networks. Events are messages that are generated when a special condition arises. The relationship between events often provides actionable insights into the cause of existing network problems as well as advanced warnings of future problem occurrences. Figure 1 illustrates event data we obtained from a production network at a large financial institution. The attributes of the data are: *Date*, *Time*, *Interval* (five minute interval), *EventType*, *Host* from which the event originated, and *Severity*. The column labeled "Rec" is only present to aid in making references to the data. Observe the following:

1. Host 23 generated a large number of InterfaceDown events on 8/21. Such situations may indicate a problem with that host.

2. When Host 45 generates an InterfaceDown event, Host 16 generates a CiscoLinkUp (failure recovery) event within the same five minute interval. Thus, a Host 45 InterfaceDown event may provide a way to anticipate the failure of Host 16.

3. The event types MLMStatusUp and CiscoDCDLinkUp tend to be generated from same Host and within the same minute. This means that when a Cisco router recovers a link, it will discover that its mid-level manager is accessible. Such event pairs should be filtered since they arise from normal operation.

4. Host 24 and Host 32 tend to generate events with same severity in the same day. This suggests a close linkage between these hosts. If this linkage is unexpected, it should be investigated to avoid having problems with one host cause problems with the other host.

Several definitions of transactions and items are needed to discover patterns (1)-(4). For (2), transactions are determined by groupings events into five minute intervals (attribute *Interval*). For (1) and (4), event groupings are done by *Date* attribute. For (3), a transaction reflects events that occur on the same Host within the same minute. The definition of items is similarly diverse. For (1) and (4), an item is a *Host*. For (3), it is an *EventType*. For (2), it is determined by the values of *Host* and *EventType*.

Herein, we extend the mining problem to include the manner in which data attributes are used to define transactions and items. One way to approach this **extended data mining problem** is to iteratively preprocess the data to form different items and transaction groupings and then apply current mining algorithms. However, this scales poorly. For example, for a data set with six attributes, it turns out that there are 665 ways to group and label records. Another approach is to mine for multi-level associations (e.g.,

(Rec)	Date	Time	Interval	EventType	Host	Severity
(1)	08/21/00	2:12am	2:10am	TcpCnnctClose	3	harmless
(2)	08/21/00	2:13am	2:10am	InterfaceDown	45	severe
(3)	08/21/00	2:14am	2:10am	InterfaceDown	23	severe
(4)	08/21/00	2:14am	2:10am	InterfaceDown	5	severe
(5)	08/21/00	2:15am	2:10am	InterfaceDown	24	severe
(6)	08/21/00	2:16am	2:15am	CiscoLinkUp	16	harmless
(7)	08/21/00	3:16am	3:15am	MLMStatusUp	16	harmless
(8)	08/21/00	3:16am	3:15am	CiscoDCDLinkUp	16	harmless
(9)	08/21/00	3:33am	3:30am	InterfaceDown	45	severe
(10)	08/21/00	3:34am	3:30am	CiscoLinkUp	16	harmless
(11)	08/21/00	3:51am	3:50am	InterfaceDown	23	severe
(12)	08/21/00	4:06am	4:05am	MLMStatusUp	19	harmless
(13)	08/21/00	4:06am	4:05am	CiscoDCDLinkUp	19	harmless
(14)	08/21/00	4:10am	4:10am	InterfaceDown	45	severe
(15)	08/21/00	4:11am	4:10am	InterfaceDown	23	severe
(16)	08/21/00	4:13am	4:10am	network down	32	severe
(17)	08/21/00	4:14am	4:10am	CiscoLinkUp	16	harmless
(18)	08/21/00	5:18am	5:15am	MLMStatusUp	19	harmless
(19)	08/21/00	5:18am	5:15am	CiscoDCDLinkUp	19	harmless
(20)	08/21/00	6:15am	6:15am	InterfaceDown	23	severe
(21)	09/15/00	3:53am	3:50am	MLMStatusUp	12	harmless
(22)	09/15/00	3:53am	3:50am	InterfaceDown	45	severe
(23)	09/15/00	3:53am	3:50am	CiscoDCDLinkUp	12	harmless
(24)	09/15/00	3:55am	2:55am	CiscoLinkUp	16	harmless
(25)	09/15/00	4:35am	4:35am	InterfaceDown	23	severe
(26)	09/15/00	5:15am	5:15am	InterfaceDown	24	severe
(27)	09/15/00	5:18am	5:15am	InterfaceDown	45	severe
(28)	09/15/00	5:19am	5:15am	SegmentDown	32	severe
(29)	09/15/00	5:12pm	5:10pm	RouterLinkDown	46	fatal
(30)	09/15/00	5:18pm	5:15pm	DBServerDown	73	fatal
(31)	09/15/00	5:21pm	5:20pm	CiscoLinkUp	16	harmless

Figure 1: System Management Events. *(Rec)* is included only for reference purposes.

[7] and [13]). Unfortunately, this requires specifying hierarchies. Since many such hierarchies are possible, considerable iteration may be necessary. Further, these approaches do not address how to group data into transactions.

1.1 Related Work

Agrawal et.al.[2, 3] identified the association rule problem and developed the level-wise search algorithm. Since then, many algorithms have been proposed to make mining more efficient (e.g. [1, 4, 8, 9] and [5] for a review). Our work builds on these efforts but broadens the scope of the mining problem.

Srikant et.al.[13] and Han et.al.[7] consider multi-level association rules based on item taxonomies, and [11] and [14] provide further extensions to handle more general constraints. All of these efforts assume that items occupy a fixed position in the hierarchy and that the hierarchies are known in advance. Further, none of these efforts considers different ways of grouping records into transactions. In contrast, our framework enables the discovery of patterns without either fixing the way in which transactions are defined or prespecifying an item hierarchy.

Also related to our work, Shen et.al.[12] develop metaqueries for Bayesian data clusters using templates expressed as second-order predicates. Fu et.al.[6] and Kamber et.al.[10] extend metaqueries to relational databases and multi-dimensional data cubes. Meta-rules can be viewed as rule templates expressed as a conjunction of predicates instantiated on a single record. In contrast, our work considers multi-attribute patterns formed from *multiple* records. Further, we consider mining the transaction groupings as well, something that the foregoing work does not address.

1.2 Contribution

The main contributions of this paper are:

1. A framework for mining multi-attribute data without prespecifying the attributes used to group records into transactions or the attributes used to define items. The framework is based on the concept of a mining camp that has three components: the pattern length (the number of items), the set of attributes used to group data into transactions, and the set of attributes used to define items.

2. The identification (and associated proofs) of two new kinds of downward closure related to searching mining camps for patterns.

3. The multi-attribute mining (*MAM*) algorithm that uses downward closure of pattern length along with the two new types of downward closure that we identify

4. Empirical studies with real world data that show a factor of three to four reduction in execution times by using the new kinds of downward closure that we identify.

1.3 Organization of the Paper

The remainder of this paper is organized as follows. Section 2.1 defines the *FARM* framework and the concepts of **mining camps** and **mining spaces**. Based on the framework, we study the downward closure properties in Section 2.2. Section 2.3 shows how the *FARM* framework can exploit predefined taxonomies. Section 3 presents the multi-attribute mining (*MAM*) algorithm that uses the downward closure properties of Section 2.2, and Section 4 compares *MAM* with *SAM*, a naive extension of the apriori algorithm. Our conclusions are contained in Section 5.

2 The *FARM* Framework

This section describes the elements of the *FARM* framework for mining data with multiple attributes. Section 2.1 provides key definitions. Section 2.2 establishes the conditions for three types of downward closure for mining within our framework. Section 2.3 extends our framework to include taxonomies and functional dependencies.

2.1 Problem Statement

The *FARM* framework goes beyond fixed attribute mining to mine directly from multi-attribute data. We are given data D with attributes $A = \{A_1, \cdots, A_k\}$. Thus, each record in D is a k tuple. For a given pattern, a subset of these attributes is used to define how transactions are grouped and another (disjoint) subset of attributes determines the items. The former are called the **grouping attributes**, and the latter are the **itemizing attributes**.

We begin with an example based on Figure 1. Here, $k = 6$. For pattern 3, the grouping attributes are $Host$ and $Time$; the itemizing attribute is $EventType$. The pattern has length two, which means that a pattern instance has two records. The items specified by these records are determined by the value of the $EventType$ attribute. That is, one record must have $EventType = MLMStatusUp$ and the other has $EventType = CiscoDCDLinkUp$. Further, these records must have the same value for their $Host$ and $Time$ attributes. Records 7 and 8 form an instance of pattern 3 with $Host = 16$ and $Time = 3:16am$. Note that items may be formed from multiple attributes. For example, pattern 2 has the itemizing attributes $Host$ and $EventType$.

We use the term **mining camp** to provide the context in which patterns are discovered. Context includes pattern length (as in existing approaches), grouping attributes, and itemizing attributes. For example, pattern 3 has the mining camp $(2, \{Host, Time\}, \{EventType\})$.

Definition 1 *A **mining camp** is a triple (n, G, S) where n is number of records in a pattern, G is a set of grouping attributes, and S is the set of itemizing attributes. A mining camp is **well formed** if $G \bigcap S = \emptyset$. A mining camp is **minable** if $S \neq \emptyset$.*

We demand that $G \bigcap S = \emptyset$ to avoid interactions between the manner in which groupings are done and items are defined. We require that $S \neq \emptyset$ since there must be items to count (even if there is only one

group).

Next, we formalize the notion of a pattern. There are several parts to this. First, note that two records occur in the same grouping if their G attributes have the same value. Let $r \in D$. We use the notation $\pi_G(r)$ to indicate the values of r that correspond to the attributes of G.

Definition 2 *Given a set of attribute G, two records r_1 and r_2 are **G-equivalent** if and only if $\pi_G(r_1) = \pi_G(r_2)$.*

In Figure 1, records 7 and 8 are G equivalent, where $G = \{Host, Time\}$.

In *FARM*, items are determined by the combinations of values of the attributes of S. Consider pattern 2 for which we require one record with $EventType = InterfaceDown$, $Host = 45$ and a second for which $EventType = CiscoLinkUp$, $Host = 16$. Thus, ($InterfaceDown$, 45) is one component (or item) of the pattern and ($CiscoLinkUp$, 16) is the other component.

Definition 3 *Given a mining camp (n, G, S) where $S = \{S_1, \cdots, S_m\}$. A **pattern component or item** is a sequence of attribute values $sv = \langle s_1, \cdots, s_m \rangle$ where $s_i \in S_i$ for $1 \leq i \leq m$. $p = \{sv_1, \cdots, sv_n\}$ is a **pattern** of length n for this mining camp if each sv_i is a pattern component for S.*

An instance of a pattern is a set of records that are in the same grouping and whose itemizing attributes match those in the pattern.

Definition 4 *Let $p = \{sv_1, \cdots, sv_n\}$ be a pattern in mining camp (n, G, S) and let D be a set of records. An **instance** of pattern p is a set of n records $R = \{r_1, \cdots, r_n\}$ such that $r_i \in D$ and $\pi_S(r_i) = sv_i$ for $1 \leq i \leq n$, and r_i and r_j are G-equivalent for all $r_i, r_j \in R$.*

Having defined what is an item (i.e., pattern component), a pattern, and a pattern instance, we now consider the support for a pattern. A G-equivalent class may have a large number of records. A decision has to be made about whether multiple instances in a G-equivalent class should provide more support than one instance. Early work [2, 3] assumes at most one pattern instance can be found in one transaction. We believe that this decision is domain dependent. So, we isolate this decision to the choice of an **aggregating function** $f : Z_+ \longrightarrow Z_+$. Two common choices of f are:

- **Existence Function**
$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$
- **Identity Function** $f(x) = x$

Now we can define the concept of support in the *FARM* framework.

Definition 5 *Given an aggregating function f, a mining camp (n, G, S) and a set of records D that can be divided to G-equivalent classes $GEC_1, \cdots, GEC_w$, the **f-support** of a pattern p is defined as $f(|GEC_1|_p) + \cdots, +f(|GEC_w|_p)$ where $|GEC_i|_p$ is the number of disjoint instances of p in GEC_i for $1 \leq i \leq w$.*

We now have in place all of the definitions necessary to discuss mining in the *FARM* framework. First, note that if G and S are fixed, then we have the traditional fixed attribute data mining problem. Here, downward closure of the pattern length is used to look for those patterns in $(n+1, G, S)$ for which there is sufficient support in (n, G, S).

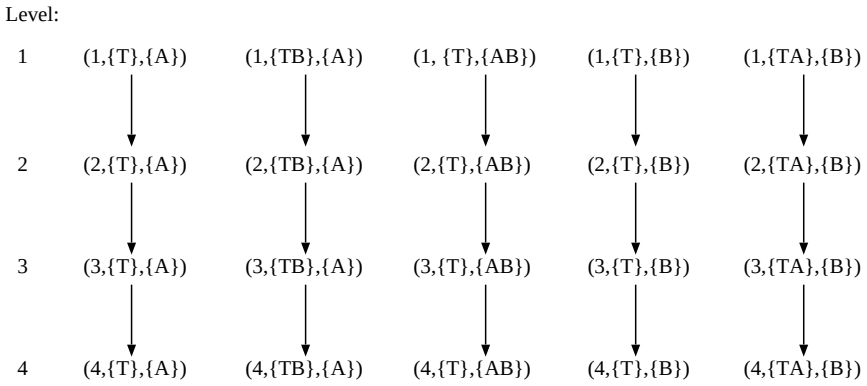


Figure 2: A Simple Search Space

In *FARM*, G and S need not be fixed. Consider the attributes T, A, B for which we require that $T \in G$. Figure 2 displays one way to search these mining camps. In essence, a separate search is done for each combination of G, S. This scales poorly. In particular, the number of permitted combinations of G and S is $3^k - 2^k$, where k is the number of attributes (which follows from observing that A_i may be in G, S, or neither and eliminating the 2^k cases for which $S = \emptyset$).

How can we efficiently search the set of possible mining camps? To provide intuition, consider Figure 1. Let $G = \{Date\}$, which results in two groups: records 1-20 and 21-31. Now consider $G' = G \bigcup \{Inteval\}$. This new set of grouping attributes refines the previous groupings. Thus, if records are not in the same $\{Date\}$ grouping, then they cannot be in the same $\{Date, Interval\}$ grouping. Hence, patterns based on these records cannot have more instance in $\{Date, Interval\}$ than they do in $\{Date\}$.

Similarly, consider $A_i \notin S$. Let p be a pattern in (n, G, S). Now consider $(n+1, G, S \bigcup \{A_i\})$. If p is a sub-pattern of p' in this second mining camp, then every occurrence of p' in this camp is also an occurrence of p in the first camp.

The foregoing suggests that mining camps can be ordered in a way that relates to downward closure.

Definition 6 *Given a mining camp $c = (n, G, S)$ and an attribute $A_i \notin G \cup S$ then*

1. *$(n+1, G, S)$ is the **type-1** successor of c.*
2. *$(n, G \cup \{A_i\}, S)$ is a **type-2** successor of c.*
3. *$(n, G, S \cup \{A_i\})$ is a **type-3** successor of c.*

Figure 3 depicts the predecessor/successor relationships present in Figure 2. The root precedes all other mining camps. (In this case, it is not a real camp since $S = \emptyset$.) The level of mining camp (n, G, S) is

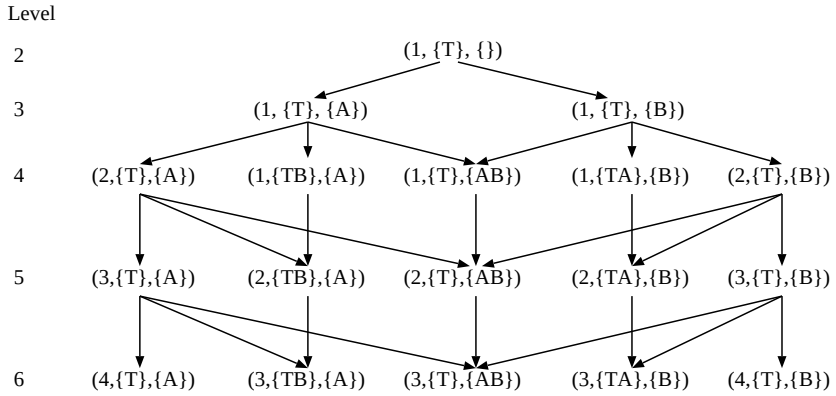


Figure 3: Search Space $MS(1, \{T\}, \{\})$ for attribute set $\{T, A, B\}$

defined as $n + |G| + |S|$. Since n is at least 1 and S is nonempty, a minable mining camp has level no less than 2. We structure the mining camps so that the successor relationships only exist between mining camps at different levels. This imposes a partial order. Figure 3 is an example of a mining space. More formally,

Definition 7 *A **mining space** $MS(c)$ is a partially ordered set (poset) of mining camps containing c and all of its successors.*

To make the notation more readable, we use $MS(n, G, S)$ to denote $MS((n, G, S))$.

The objective of the *FARM* framework is now clear

Definition 8 *A* FARM *problem is a triple $(MS(c), f, \theta)$ where f is an aggregating function and θ is the threshold. The solution of a* FARM *problem in dataset D is all patterns of every mining camp in $MS(c)$ with f-support greater than θ.*

One concern with this problem formulation is the potential for an explosive growth in the number of mining camps as the number of attributes increases. Many of these mining camps may contain meaningless combinations of itemizing and/or grouping attributes. This problem can be addressed, in part, by employing a rule-based mechanism that allows domain experts to specify the part of the mining space that may contains interesting patterns. In particular, such user-defined directives could be expressed as predicates on the elements in G and S, such as which attributes can be members of which set and under what conditions (e.g., always, never, only if another attribute is not present). We note, however, that removing some mining camps from a mining space does not necessarily guarantee faster execution because the results of removed mining camps may be used to reduce the number of candidates of the next level.

2.2 Downward Closure Properties

This section shows that several types of downward closure can be present in the *FARM* framework. Exploiting these properties provides considerable benefit in terms of efficiency.

We begin by defining properties of the aggregating function.

Definition 9 *Assume f is an aggregating function, then*

1. *f is type-1 downward closed if f is non-decreasing.*
2. *f is type-2 downward closed if f is monotonic increasing and for any two G-equivalent classes GEC_1 and GEC_2 and a given pattern p, $f(|GEC_1|_p) + f(|GEC_2|_p) \leq f(|GEC_1 \cup GEC_2|_p)$.*
3. *f is type-3 downward closed if f is non-decreasing.*

Note that by this definition, f is type-1 downward closed if and only if f is type-3 downward closed. Our main result is that downward closure is possible for n, G, and S.

Theorem 1 *Given a mining camp $c = (n, G, S)$ and an aggregating function f such that the f-support of a pattern $p = \{sv_1, \cdots, sv_n\}$ is less than θ.*

1. *If f is type-1 downward closed then for any type-1 successor of c, any pattern that is a superset of p has f-support less than θ.*
2. *If f is type-2 downward closed then the f-support of p in any of type-2 successor of c is less than θ.*
3. *If f is type-3 downward closed then the f-support of pattern $p' = \{sv'_1, \cdots, sv'_n\}$ of any type-3 successor of c is less then θ if $sv_i \subset sv'_i$ for all $1 \leq i \leq n$.*

Proof:

1. This is the a priori property proved in [3].
2. For A_i not in $G \cup S$, let $G' = G \cup \{A_i\}$ and consider $c' = (n, G', S)$, a type-2 successor of c. Let $GEC_1, \cdots, GEC_m$ be the G-equivalent classes in c, and $GEC'_1, \cdots, GEC'_{m'}$ be the G'-equivalent classes in c'. Note that for all $1 \leq j \leq m$, there is a set $Z_j \subset \{GEC'_1, \cdots, GEC'_{m'}\}$ and $GEC_j = \bigcup_{V \in Z_j} V$. Now, consider a pattern p. Its support in G is θ_p and its support in G' is θ'_p . Observe that

$$\begin{aligned} \theta_p &= \sum_{j=1}^{m} f(|GEC_j|p) \\ &= \sum_{j=1}^{m} f(|\bigcup_{V \in Z_j} V|p) \\ &\geq \sum_{j=1}^{m} \sum_{V \in Z_j} f(|V|p) \\ &= \theta'_p \end{aligned}$$

   The inequality holds because of f being type-2 downward closed.

3. Suppose $c' = (n, G, S')$ is a type-3 successor of c. The G-equivalent classes of c remain intact in c'. It is obvious that for every pattern instance $inst' = \{sv'_1, \cdots, sv'_n\}$, its projection to S, $inst = \{\pi_S(sv'_1), \cdots, \pi_S(sv'_n)\}$ is a pattern instance of c. But not every pattern instance of c can expand to a pattern instance of c'. So the f-support of $inst'$ is lower than that of $inst$.

Downward closure properties are the foundation of *MAM* as they are in traditional (fixed attribute) mining for frequent itemsets. The more downward properties the chosen aggregating function has, the greater the efficiencies that can be realized in mining. Note that the identity function has all three downward closure properties. However, the existence function is type-1 and type-3 downward closed but not type-2 downward closed.

2.3 Taxonomies and Functional Dependency

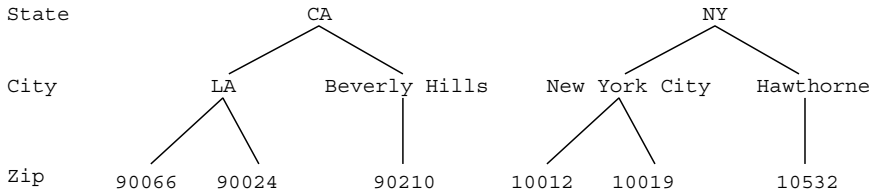


Figure 4: Example of a Taxonomy

In some situations, taxonomies (*is-a* hierarchies) are available. For example, Figure 4 shows a taxonomy of geographical information with three levels: *zip code*, *city* and *state*. A reasonable database design is to store only the lowest level attribute, e.g. *zip code*, in the main table, keep the taxonomies in a separate table, and create a logical view that contains all attributes for data ming.

Since the value of a lower level attribute uniquely determines the value of attributes in higher level, taxonomies are special cases of functional dependencies. So it is sufficient to discuss functional dependencies.

As previously stated, the number of mining camps grows exponentially with the number of attributes. There is definitely no need to discover that "houses located in the same zip code tend to be in the same city". The following theorem shows how to avoid such unnecessary computation.

Theorem 2 *Suppose U,V, G and S are attribute sets and U uniquely determine V.*

1. *The output of $(n, U \cup V \cup G, S)$ and $(n, U \cup G, S)$ are identical.*
2. *The output of $(n, G, S \cup U \cup V)$ can be derived from the output of $(n, G, S \cup U)$ by looking up the taxonomy.*
3. *For $n > 1$, $(n, U \cup G, V)$ has no pattern.*

Proof:

1. In each $U \cup G$-equivalent class, every record is also V-equivalent. That is, $U \cup G$-equivalent classes are exactly $U \cup V \cup G$-equivalent classes. So the two camps have the same grouping and labeling hence produce identical result.
2. trivial.

3. Let $G' = G \bigcup U$, and $c' = (n, G', V)$. Now consider r_1 and r_2 in a G'-equivalent class. Clearly, $\pi_U(r_1) = \pi_U(r_2)$. But since U determines V, we have $\pi_V(r_1) = \pi_V(r_2)$. Thus, there can be at most one distinct item in each G'-equivalent class.

All three properties are useful in making mining more efficient. Property (1) allows us to skip certain attribute combinations. Property (2) provides a more efficient means to determine the patterns in a mining camp. Property (3) allows us to truncate a search early.

3 Algorithm

This section describes the **multiple attribute mining** (*MAM*) algorithm for mining *FARM* problems. *MAM* exploits the downward closure properties in Theorem 1 to improve the efficiency of mining.

The extended mining problem we address raises some difficult scaling issues as a result of discovering mining camps with different grouping attributes (G). Existing mining algorithms assume that data are sorted by transaction identifier so that locality can be exploited in counting pattern instances. Such locality can be imposed on *FARM* problems as well if there is an attribute T, called the **ordering attribute** such that: (1) T is required to be in G, (2) data records are sorted by T, and (3) all of the records in a T-equivalent class fit in main memory. Possible ordering attributes include those that deal with time (e.g., day) and place (e.g., zip code). However, even if locality is not present, other techniques can be used to improve efficiency, such as decomposing the problem into subproblems with fewer attributes. In the sequel, we assume there is an ordering attribute T, and we address the mining problem $(MS(1, \{T\}, \emptyset), f, \theta)$.

We describe *MAM* in an object-oriented fashion. The core data structure is the *Camp* class.

```
class  Camp
{
    n:          Integer                  // number of items
    G:          Sequence of Attributes   // grouping attributes
    S:          Sequence of Attributes   // surrogate attributes
    pred1:      Camp                     // type 1 predecessor
    pred2:      Set of Camp              // type 2 predecessors
    pred3:      Set of Camp              // type 3 predecessors
    ptrns:      Set of Pattern           // candidates and patterns
}
```

The member *ptrns* contains candidates before counting their f-support has been computed, and it contains patterns after counting is completed and low-support candidates are removed. The *Pattern* class is defined as:

```
class  Pattern
{
    svalue:     Set of Sequences of Attribute Values   // See Definition 3
    support:    Real                                   // f-support
}
```

The *MAM* algorithm adapts to the choice of the aggregating functions. If the aggregating function has all three downward closure properties, the mining space looks like Figure 3, and the lowest level containing minable camps is level 3, e.g. $(1, \{T\}, \{A_i\})$. Otherwise, the mining space looks like Figure 2 in which the level of a mining camp is defined as the number of items, n, in the camp.

We structure the *MAM* algorithm into seven routines. Algorithm 1, the top level routine, is very similar to the classical apriori algorithm. However, Algorithm 1 sets levels based on the kinds of downward closure present, and the algorithm operates on mining camps, not candidate patterns.

Algorithm 2, CampGen, is called by Algorithm 1 to generate mining camps. Note that type-2 downward closure is used to make camp generation more efficient.

Algorithm 3, SetPredAndCandiGen, determines the predecessor to use when extending the set of patterns. Type-2 downward closure is exploited here as well.

Algorithm 1 MAIN(AttributeSet: A, Ordering-Attribute: T, Dataset: D, MinSupport: θ)

Input: $A = \{A_1, \cdots, A_k\}$: set of attributes.
T: a special ordering attribute
D: a dataset with $k+1$ attributes $(A \cup \{T\})$, sorted by attribute T.
θ: minsupport threshold
Output: frequent itemsets of all the mining camps

if f is type-2 downward closed **then**
  $l \leftarrow 3$
else
  $l \leftarrow 1$
end if
$camps_l \leftarrow CampGen(l, A)$
for each $camp \in camps_l$ **do**
  $camp.ptrns \leftarrow$ {frequent 1-itemsets }
end for
while exist $camp \in camps_l, camp.ptrns \neq \emptyset$ **do**
  Evaluate$(D, camps_l)$
  for each $camp \in camps_l$ **do**
    eliminate patterns with support lower than θ in $camp.ptrns$;
  end for
  $l \leftarrow l + 1$
  $camps_l \leftarrow CampGen(l, A)$
  $camps_l.SetPredAndCandiGen()$
end while
return $\{camp.ptrn | camp \in camps_l, l \geq 3\}$

Algorithm 4, CandiGen, applies the extended downward closure properties. There are two issues here: how to generate candidates and how to filter out impossible candidates. The initial candidate set can be generated from the pattern sets of any of type of predecessors. But in general, the most efficient candidate generation is to start from patterns of type-2 predecessors. This is because patterns of type-2 predecessors

Algorithm 2 CampGen(Level: l, AttributeSet: A, Ordering-Attribute: T)

if f is type-2 downward closed **then**
  $camps \leftarrow \{(n, \{T\} \cup G, S) | 1 \leq n \leq l-2, G \subset A, |G| \leq l-n-2, S \subseteq A-G, |S| = l-n-|G|-1\}$
else
  $camps \leftarrow \{(l, \{T\} \cup G, S) | G \subset A, S \subseteq A-G, S \neq \emptyset\}$
end if
return *camps*

Algorithm 3 SetPredAndCandiGen(CampSet: *camps*)

for each $camp = (n, G, S) \in camps$ **do**
  if $n > 1$ **then**
    $camp.pred1 \leftarrow (n-1, G, S)$
  end if
  if $|S| > 1$ **then**
    $camp.pred3 \leftarrow \{(n, G, S - \{A_y\}) | A_y \in S\}$
  end if
  if (f is type-2 downward closed) and ($|G| > 1$) **then**
    $camp.pred2 \leftarrow \{(n, G - \{A_x\}, S) | A_x \in G, A_x \neq T\}$
  end if
  $CandiGen(camp)$
end for

have the same n and S as their successors. Thus, successor patterns are computed by refining the G-equivalent classes of the predecessor. This is done by taking intersections, a computation that can be done in linear time (if patterns are sorted). In contrast, both type-1 and type-3 require a "join" operation. Not not only is this more computationally intensive, the number of candidates generated tends to be very large. The algorithm tries to generate candidates from the pattern sets of its type-2 predecessors and then uses pattern sets of other types to further filter candidates. If there is no type-2 predecessor, the type-1 predecessor is used instead. If there is no type-1 predecessor, type-3 predecessors are used.

Algorithm 5, Evaluate, computes the support level of candidate patterns. Each pattern component is checked in turn. The resulting support level is the minimum of f applied to the minimum of the count of each pattern component.

Algorithm 6, AttrHash, provides a hash table to the counting routines.

Algorithm 7, PatternComponentCount, builds the count matrix for each pattern component. A pattern component is a set of $|S|$ attribute values, and it is 'satisfied' by a tuple if all the values appear in the corresponding attributes of the tuple. We use an array Sat_{pc} to store the number of attribute values satisfied by the current tuple. For each attribute value a_k in the tuple, we retrieve all the pattern components that has constraint $A_k = a_k$ from the hash table for attribute k, and increase the Sat count of the pattern component by 1. A pattern component is satisfied by the tuple if all of its constraints are satisfied, and support of the pattern component is increased by 1.

One remaining issue is how to choose a set of mining camps to be mined in a pass of data scan. A very

Algorithm 4 CandiGen(Camp: *camp*)

if $camp.pred2 \neq \emptyset$ **then**
  $camp.ptrns \leftarrow \bigcap_{c \in camp.pred2} c.ptrns$
  Type1Filter($camp.ptrns, camp.pred1$)
  Type3Filter($camp, camp.pred3$)
else if $camp.pred1 \neq \emptyset$ **then**
  $camp.ptrns \leftarrow \{p = p_1 \cup p_2 | \{p_1, p_2\} \subset camp.pred1.ptrns, |p| = n\}$
  Type1Filter($camp.ptrns, camp.pred1$)
  Type3Filter($camp, camp.pred3$)
else
  $c_1 \leftarrow$ the camp with the least number of patterns in $camp.pred3$
  $c_2 \leftarrow$ the camp with the least number of patterns in $camp.pred3 - \{c_1\}$
  $camp.ptrns \leftarrow \{\langle s_1, \cdots, s_n \rangle | \langle \pi_{c_1.S}(s_1), \cdots, \pi_{c_1.S}(s_n) \rangle \in c1.ptrns,$
    $\langle \pi_{c_2.S}(s_1), \cdots, \pi_{c_2.S}(s_n) \rangle \in c2.ptrns$ and $|s_i| = |S|$, for $1 \leq i \leq n\}$
  Type3Filter($camp, camp.pred3 - \{c_1, c_2\}$)
end if
return

SUBROUTINE Type1Filter(*ptrns*, *camp*)
if $camp \neq null$ **then**
  for each $p \in ptrns$ **do**
    for each $p' \subset p$, $|p'| = n - 1$ **do**
      eliminate p from $ptrns$ if $p' \notin camp.ptrns$
    end for
  end for
end if

SUBROUTINE Type3Filter(*thiscamp*, *camps*)
if $camps \neq \emptyset$ **then**
  for each $p \in thiscamp.ptrns$ **do**
    for each $S' \subset thiscamp.S$ where $|S'| = |S| - 1$ **do**
      eliminate p from $ptrns$ if there exists $c \in camps$ such that $c.S = S'$ and $\pi_{S'}(p) \notin c.ptrns$
    end for
  end for
end if

Algorithm 5 Evaluate(Dataset: D, CampSet: $Camps$)

$H \leftarrow AttrHash(Camps)$
$p.support \leftarrow 0$ for all pattern p
while not End Of File **do**
  $TD \leftarrow ReadBlock(D)$
  $count \leftarrow PatternComponentCount(D, H)$
  for each $camp = (n, G, S) \in Camps$ **do**
    for each $p \in camp.ptrns$ **do**
      $p.support \leftarrow p.support + f(min(\{count_{g,pc} | pc \in p, g is an instance of G\}))$
    end for
  end for
end while

Algorithm 6 AttrHash(CampSet: $camps$)

for each attribute k **do**
  $H_k \leftarrow newHash()$;
end for
for each pattern p_{ij} that appears in camp c_i **do**
  for each pattern component $pc \in p_{ij}$ **do**
    pc is in the form of $\langle A_1 = a_1, \cdots, A_k = a_k, \cdots \rangle$;
    insert pair (a_k, pc) into hash table H_k;
  end for
end for

natural design, as adopted in *MAM*, is to mine camps on same level in one data scan because each camp has to wait for the result of camps in the previous level. This design is reflected in Algorithm 1 and Algorithm 5.

We conclude this section by mentioning some additional efficiencies that can be obtained. First, note that if the aggregating function is type-2 downward closed, the patterns of $(1, \{T\} \cup G, S)$ and $(1, \{T\}, S)$ are identical because the number of one-item instances is not affected by grouping. Also, observe that if a mining camp has a predecessor of any type with no pattern, the camp has no pattern either. This is a direct result of the downward closure property.

4 Performance

This section assesses the performance of the *MAM* algorithm. Section 4.1 details the data used in our assessments. Section 4.2 compares the *MAM* algorithm with one that does not exploit downward closure for G (type 2) or S (type 3). The results show that *MAM* provides considerable efficiencies, reducing execution times by a factor of three to four.

4.1 Data

Two kinds of data are used in our study. The first are event data taken from a production computer network at a financial service company. One data set (NETVIEW) has six attributes: *Hour*, *EventType*, *Host*,

Algorithm 7 PatternComponentCount(DataBlock D, HashArray H)

```
for each tuple r = ⟨a_1, ··· , a_n⟩ ∈ D do
  Sat_pc ← 0, for all pattern component pc;
  for each attribute k do
    for each pc retrieved by key a_k in hash table H_k do
      pc is a pattern component in the patterns of camp = (n, G, S);
      Sat_pc ← Sat_pc + 1;
      if Sat_pc = |S| then
        g ← π_G(r);
        count_{g,pc} ← count_{g,pc} + 1;
      end if
    end for
  end for
end for
```

Severity, *Interestingness*, *DayofWeek*; *Hour* is used as the ordering attribute. There are 241 values of *EventType*, 2526 for *Host*, 5 levels of *Severity*, and 5 for *Interestingness*. The second data set (TEC) has the attributes *Hour*, *EventType*, *Source*, *Severity*, *Host* and *DayOfYear*. Again, *Hour* is used as the ordering attribute. There are 75 values of *EventType*, 16 types of *Source*, 2718 *Host* values, and 7 *Severity* levels.

We also generated synthetic data (SYN) to cover a large range of data characteristics. Data generation is parameterized to take into account many factors, such as the number of records, the number of attributes, and the pattern lengths. The generation of discrete random variables uses the Poisson distribution.

4.2 Comparisons

To our best knowledge, there is no existing algorithm that solves the mining problem proposed in this paper. A naive approach is to preprocess the data to obtain every possible combination of G and S. Clearly, this has poor efficiency since it requires many data scans.

A more insightful study is to compare *MAM* with an approach that only uses downward closure based on pattern length and ignores the type-2 and type-3 downward closure. We refer to this as the *Single Attribute Mining* algorithm or *SAM*. *SAM* is a degenerate case of *MAM* in which the aggregating function is not type-2 nor type-3 downward closed[1]. Put differently, we show the performance gain obtained by exploiting type-2 and type-3 downward closure.

Figure 5 compares *MAM* and *SAM* in terms of the number of the number of candidates generated and execution time. The rationale for doing so is that *MAM* and *SAM* require about same number of data scans. So, the reduced execution time achieved by *MAM* is entirely due to having far fewer candidates. In our experiments, both real world data sets and the synthetic data show similar result. *SAM* generates several

[1]The *MAM* algorithm shown in Section 3 is based on the assumption that if an aggregating function is type-1 downward closed, then it is also type-3 downward closed. It is not too difficult to modify the algorithm so only the type-1 downward closure property is used.

orders of magnitude more candidates than *MAM*. As a result, *SAM* execution times are often a factor of three to four greater than *MAM* execution times.

A closer look at this figure reveals that the *MAM* provides the greatest speedup over *SAM* when there is the *smallest* difference in the number of candidates generated. This occurs when the support threshold is smallest. To understand why, we must dig deeper. The overhead at the first level of the algorithms does not depend on the support threshold. For *SAM*, the overhead is proportional to the number of distinct records; for *MAM*, the overhead is proportional to the sum of number of distinct values of each attribute. In both cases, the number of candidates considered is fixed. But at low support thresholds, processing at the first level dominates the overall execution time.

In Figure 6, we show that the execution times of *MAM* and *SAM* are roughly linear to the number of records, although execution times do increase as the support threshold decreases. However, *SAM* execution times increase at a considerably faster rate than those for *MAM*.

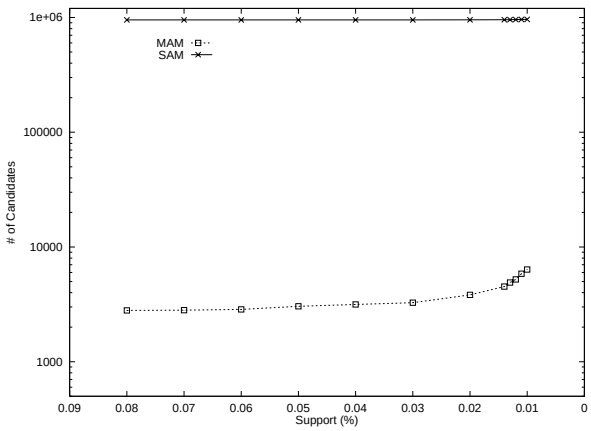
We conclude this section by listing some of the patterns discovered by *MAM* in the NETVIEW data. The camp $(2, \{Hour\}, \{Host, Severity, Interestingness\})$ generates output (⟨Host3,Harmless, VeryInteresting⟩, ⟨Host17,minor warning, some interest⟩) with 1.12% support. This suggests that *Host17* has some close relationship (potential causal) with key events that occur on *Host3*. Such insights provide an excellent starting point for problem determination. Another such insight is obtained from the camp $(3, \{Hour, DayOfWeek\}, \{Host\})$, which generates the pattern $\{\langle Host3\rangle, \langle Host17\rangle, \langle Host31\rangle\}$ with support 1.01%.

5 Conclusion and Future Work

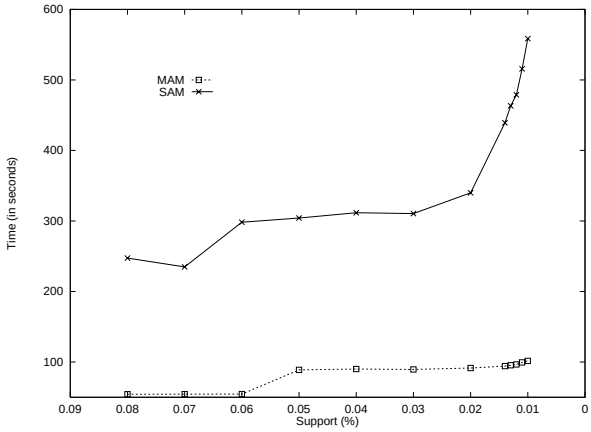
Mining typically involves a preprocessing step in which data with multiple attributes are grouped into transactions, and items are defined based on attribute values. Unfortunately, fixing the attributes used to define transactions and items can severely constrain the patterns that are discovered.

This has motivated us to introduce *FARM*, a new framework for mining multi-attribute data. In *FARM*, mining is done directly from multi-attribute data without prespecifying the attributes used to group records into transactions or the attributes used to define items. The framework is based on the concept of a mining camp: (n, G, S), where n is the pattern length (the number of items), G is the set of attributes used to group data into transactions, and S are the attributes used to define items. We identify (and prove) two new kinds of downward closure related to searching mining camps for patterns based on the relationship between the G and S. These results are incorporated into the multi-attribute mining *MAM* algorithm. Empirical studies with real world data show a factor of three to four reduction in execution time by using the new kinds of downward closure that we identify.

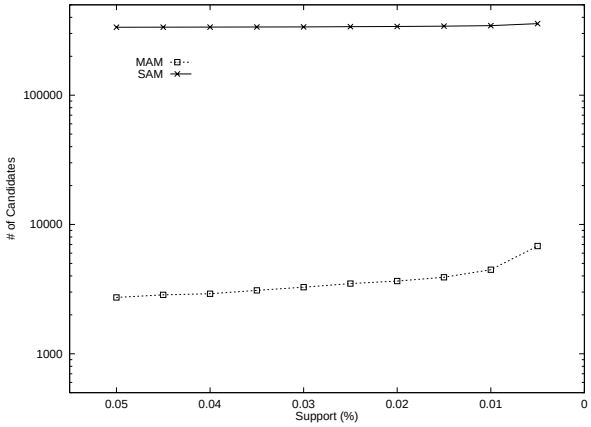
Much work remains. In particular, we hope to extend the framework to include numeric attributes. We are also interested in irregular mining camps, camps that have different itemizing attributes for each record. Last, we want to allow patterns to contain wildcard values.



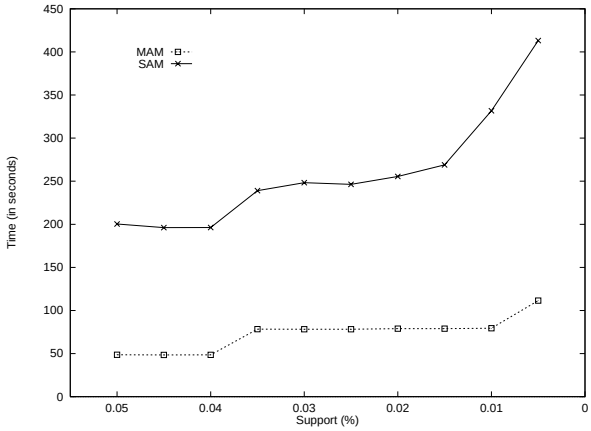
(a) TEC: number of candidates



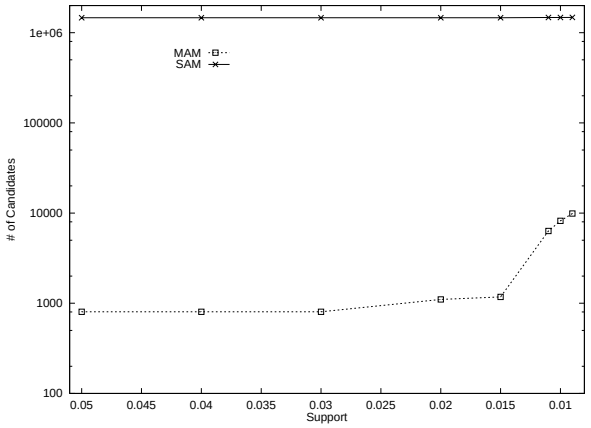
(b) TEC: execution time



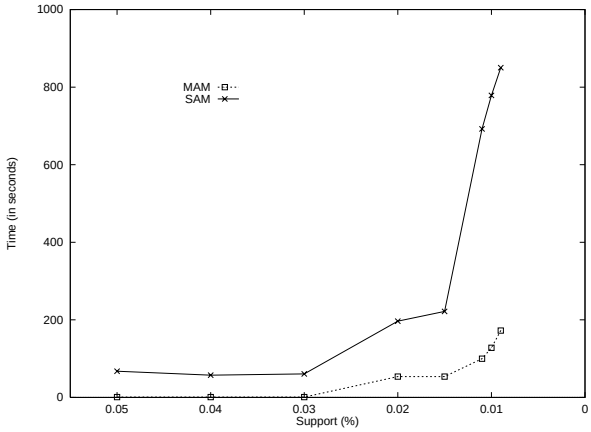
(c) NETVIEW: number of candidates



(d) NETVIEW: execution time

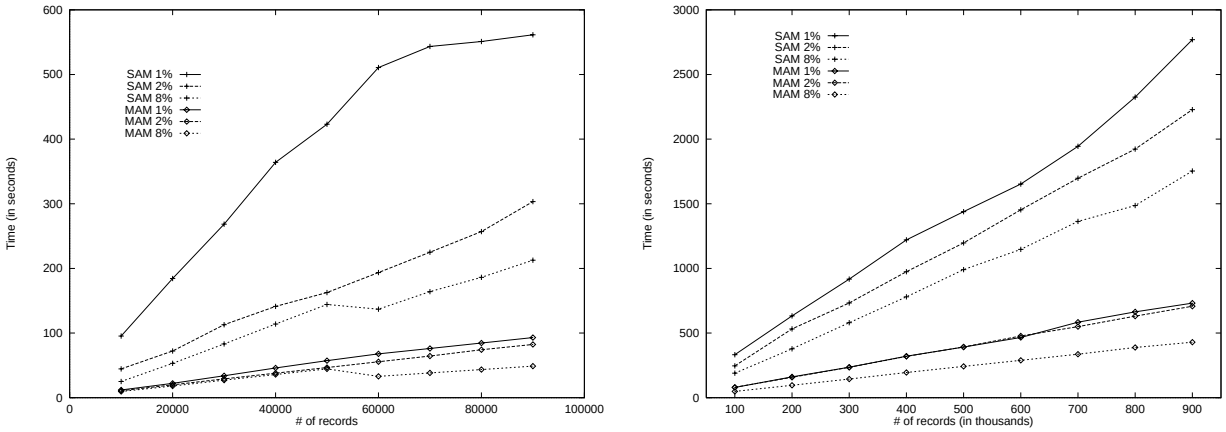


(e) SYN: number of candidates



(f) SYN: execution time

Figure 5: Performance Comparison



(a) TEC (b) NETVIEW

Figure 6: Effect of the Number of Records at Different Support Levels

References

[1] Aggarwal, C., Aggarwal, C., and Parsad, V. Depth first generation of long patterns. In *Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)* (2000).

[2] Agrawal, R., Imielinski, T., and Swami, A. Mining association rules between sets of items in large databases. In *Proc. of Very Large Database (VLDB)* (1993), pp. 207–216.

[3] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. In *Proc. of Very Large Database (VLDB)* (1994).

[4] Bayardo, R. Efficiently mining long patterns from database. In *SIGMOD* (1998), pp. 85–93.

[5] Deogun, J., Raghavan, V., Sarkar, A., and Sever, H. Data mining: Research trends, challenges, and applications, 1997.

[6] Fu, Y., and Han, J. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int'l Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95), Singapore.* (1995), pp. 39–46.

[7] Han, J., and Fu, Y. Discovery of multiple-level association rules from large databases. In *Proc. of Very Large Database (VLDB)* (1995).

[8] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In *Int. Conf. Management of Data (SIGMOD)* (2000).

[9] Hipp, J., Myka, A., Wirth, R., and Guntzer, U. A new algorithm for faster mining of generalized association rules. In *Proc. 2nd PKKD, 1998.* (1998).

[10] Kamber, M., Han, J., and Chiang, J. Y. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)* (1997), pp. 207–210.

[11] Ng, R., Lakshmanan, L., Han, J., and Pang, A. Exploratory mining and pruning optimizations of constrained associations rules. In *Int. Conf. Management of Data (SIGMOD)* (1998), pp. 13–24.

[12] Shen, W., Ong, K., Mitbander, B., and Zaniolo, C. *Metaqueries for data mining.* AAAI/MIT press, 1996, pp. 375–398.

[13] Srikant, R., and Agrawal, R. Mining generalized association rules. In *Proc. of Very Large Database (VLDB)* (1995), pp. 407–419.

[14] Srikant, R., Vu, Q., and Agrawal, R. Mining association rules with item constraints. In *Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)* (1997), pp. 67–93.