Discovery in Multi-Attribute Data with User-defined Constraints

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

There has been a growing interest in mining frequent itemsets in relational data with multiple attributes. A key step in this approach is to select a set of attributes that group data into transactions and a separate set of attributes that labels data into items. Unsupervised and unrestricted mining, however, is stymied by the combinatorial complexity and the quantity of patterns as the number of attributes grows. In this paper, we focus on leveraging the semantics of the underlying data for mining frequent itemsets. For instance, there are usually taxonomies in the data schema and functional dependencies among the attributes. Domain knowledge and user preferences often have the potential to significantly reduce the exponentially growing mining space. These observations motivate the design of a user-directed data mining framework that allows such domain knowledge to guide the mining process and control the mining strategy. We show examples of tremendous reduction in computation by using domain knowledge in mining relational data with multiple attributes.

Keywords

domain knowledge,frequent itemset, association rule, multi-attribute

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 `MarketBasket` attribute might be used to group data into transactions and the `ProductType` attribute (with values such as `domestic 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. Unfortunately, fixed attribute mining imposes severe limitations on the patterns that can be discovered in that the analyst must specify in advance the attributes used to designate items (e.g., `Imported/Domestic` and `Product Class`) and how to group them for the purposes of pattern discovery (e.g., `Transaction` or `CustomerName` + `TimeOfDay`). To this end, a framework for mining multi-attribute data, *FARM*[13], proposes an approach to mining multi-attribute data in which itemizing and grouping attributes are selected in the course of the mining operation itself. While this greatly expands the range of patterns that can be discovered, it also creates another level of combinatorial complexity. This paper proposes a framework for specifying constraints on the itemizing and grouping attributes. Not only does this reduce computational complexity, it can also result in patterns of more interest to the analyst.

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, **multiple-attribute mining** applies the notion of mining spaces to discover frequent patterns for transactions and items that are defined in terms of data attributes. A **transaction** is a general term for a group of records. This approach does not require pre-specified taxonomies, although it exploits such information if it is available. Further, because of various downward closure properties, multiple-attribute mining is considerably faster than simply employing apriori-like algorithms on each choice of attributes for defining transactions and items.

To illustrate the foregoing and to better motivate the problem we address, consider the domain of event management of complex networks. Events are messages that are generated when special conditions arise. The relationship between events often provides actionable insights into the cause of current network problems as well as advanced warnings of future problem occurrences. Figure 1 illustrates a portion of event data we obtained from a production network at a large financial institution. We initially focus on the attributes `Date`, `Time`, `Intrvl5` (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. The full data set evidenced the following patterns:

1. Host `netsvr38` generated a large number of `IntrfcDwn` events on 8/21. Such situations may indicate a problem with that host.

2. When host `netsvr22` generates an `IntrfcDwn` event, host `router16` generates a `CiscoLnkUp` (failure recovery) event within the same five minute interval. Thus,

Rec	Date	Time	AM/PM	Intrvl5	Intrvl30	EventType	Host	Site	Source	Subsource	Severity	Maint
(1)	8/21	2:12:23	AM	2:10	2:00	TcpCls	prtsvr3	haw	infoprint	prtdaemon	harmless	No
(2)	8/21	2:13:41	AM	2:10	2:00	IntrfcDwn	netsvr38	ykt	netagt	cat5-agt	severe	No
(3)	8/21	2:14:11	AM	2:10	2:00	IntrfcDwn	netsvr22	haw	netagt	cdl-agt	severe	No
(4)	8/21	2:14:37	AM	2:10	2:00	IntrfcDwn	netsvr5	haw	netagt	ibm-agt	severe	No
(5)	8/21	2:15:02	AM	2:15	2:00	IntrfcDwn	netsevr24	haw	netagt	ibm-agt	severe	No
(6)	8/21	2:16:09	AM	2:15	2:00	CiscoLnkUp	router16	haw	cisco-agt	cat5-agt	severe	Yes
(7)	8/21	2:38:48	AM	2:35	2:30	NetMgrUp	netview16	ykt	netview	ibm-nev	harmless	No
(8)	8/21	2:48:23	AM	2:45	2:30	RtrLnkUp	router16	haw	cisco-agt	cat5-agt	harmless	No
(9)	8/21	3:13:12	AM	3:10	3:00	IntrfcDwn	netsrv45	ykt	netagt	tvl-agt	severe	No

Figure 1: Distributed System Management Events

an `IntrfcDwn` on `netsvr22` may provide a way to anticipate the failure of `router16`.

3. The event types `MLMStatusUp` and `CiscoDCDLnkUp` 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. Hosts `netsvr38` and `netsvr5` 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 `Intrvl5`). 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`.

While *FARM* discovers a more complete set of patterns, it creates challenges as well. First, analysts may be overwhelmed by dealing with the abundance of patterns discovered. Second, even though the *FARM* approach exploits downward closure properties to provide computational efficiencies, the time required for pattern discovery can be substantial. For example, we show later in the paper that mining data with 20 attributes is equivalent to performing $3,485,735,825$ (or $3^{20} - 2^{20}$) rounds of market-basket style mining on the same data set if all different combinations of itemizing and groupings are to be explored. Thus, it is clear that such unsupervised approach is not feasible.

The foregoing motivates us to constrain the selection of grouping and itemizing attributes so as to make *FARM* more computationally efficient and its results more meaningful. To this end, we develop attribute predicates that constrain the ways in which grouping and itemizing occur, and we show how these predicates can be incorporated into *FARM*. Figure 1 provides examples of such predicates, especially if we also consider the attributes `AM/PM` (twelve-hour period), `Intrvl30` (thirty-minute intervals), `Site` (location of the host), and `Maint`.

1. `Time` is in a twelve hour format. Thus, if the grouping attributes include `Time`, they should also include `AM/PM`. The same argument applies to `Intrvl5` and `Intrvl30`.

2. The reason for a perceived host failure may be that it is recovering after a normal maintenance operation. Indeed, in Figure 1 we see that the failure of `router16` at `2:16:09` occurs within the maintenance window for that host. Thus, if `EventType` is an itemizing attribute, we should also include `Maint`.

3. Certain logical dependencies exist in the data that can reduce the attribute combinations. For example, if we use `Intrvl5`, we know `Intrvl30` (i.e., there is a functional dependency). Similarly, if we know the `Host`, then we know the `Site`.

Exploiting these relationships between attributes can result in a substantial reduction in the number of patterns reported. Indeed, in our studies, reductions of several orders of magnitude are achieved.

1.1 Problem Statement and Scope

We have two goals in this study. The first is to design a small and comprehensible set of directives that allow users to specify the domain knowledge based on attributes in an intuitive way. The specification language should be expressive enough to incorporate common knowledge without operational instructions. For example, users should be able to indicate relationships (such as functional dependencies) among attributes, in which way should these attributes be used in itemizing or grouping, and whether some attributes should be included in mining at all.

The second goal is to design an inference system that can translate the domain knowledge expressed in this denotational specification language to operational instructions that guide the frequent itemset mining algorithms to avoid unnecessary computation.

To realize these goals, we organize the search space into novel architecture that is conducive to attribute-based pruning. Our approach is based on the FARM framework [13], where each mining template directly corresponds to a unique attribute mapping, and connects to other mining templates through a rich set of downward closure relationships. It is through these relationships that user-directed pruning of the search space takes place. In this paper, we omit issues such as candidate generation, aggregating functions, and instance counting. Interested readers can find the details in [13].

1.2 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; 9; 10] and [5] for a review). Our work builds on these efforts but broadens the scope of the mining problem.

Mining data with multiple attributes has been recognized as an important task. Srikant et.al. [16] and Han et.al. [8] consider multi-level association rules based on item taxonomies and hierarchies. More recently, Grahne et.al. [7] proposed the dual mining for mining situations of a frequent pattern. Our previous work [13] extends these work under a more general setting. It studied how different mining tasks with different attributes mappings are related and thus allows us to reduce search space drastically. The work presented in this paper has a very different focus in that we seek effective ways to express knowledge, and we develop an efficient algorithm that leverage the knowledge for mining process optimization.

Considerable work has been done in characterizing pattern interestingness [15; 12] and item constraints [11; 17]. Such interestingness and constraints are defined based on items (e.g. item a and b should not appear in the same pattern). A framework has been developed for describing either interestingness or constraints on items and for efficient mining by pushing the constraints to the level-wise search in reducing the number of candidates generated at each level. In contrast, this paper discusses how to express knowledge in a much more general way (e.g. attributes {event, type, name} can not used as itemizing attributes at the same time), which is on a higher level than the item-based approach. We demonstrate that such knowledge (or constraints) about variables help us to drastically reduce the mining space. Further, we describe a language that can be used to describe common constraints and develop algorithms to construct optimal mining paths.

1.3 Paper Organization

We first review the *FARM* framework in Section 2. In Section 3, we describe the Attribute Specification Language (ASL), which enables users to specify attribute-level domain knowledge. Section 4 describes how the system interprets the specifications in ASL and translates them to data mining constraints. Section 4.1 introduces the concept and realization of semantic closure. Sections *4.2.1*, *4.2.2*, *4.2.3* and *4.2.4* present various types of mining camps that can be pruned from mining spaces and the methods of identifying them. Section *4.2.5* describes the properties of the inference methods and how it can be implemented as a constraining module of *FARM.* In Section 5, we demonstrate how to express domain knowledge in ASL and how the inference system can greatly reduce the mining space. Section 6 concludes the paper.

2. THE FARM FRAMEWORK

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. Such fixed attribute mining can severely limit the patterns that are discovered. For instance, 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 define items. Fixing the attributes used to define transactions and items can severely constrain the patterns that are discovered. For example, by defining transactions as products purchased together, we may fail to discover relationships between items purchased by the same family in a single month.

To go beyond fixed attribute mining to mine directly from multi-attribute data, we introduced *FARM* [13]. In *FARM*, *transaction* is a general term for a group of records, which are formed dynamically based on the values of a set of attributes. More formally, we are given data D with attributes $A = \{A_1, \cdots, A_k\}$. 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.

We use the term **mining camp** to provide the context in which patterns are discovered. The context includes the length of the pattern (as in existing approaches), the grouping attributes, and the itemizing attributes.

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, S is a set of itemizing attributes, and $G \bigcap S = \emptyset$, $S \neq \emptyset$.*

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 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 $1 \leq i \leq m$. We call $p = \{sv_1, \cdots, sv_n\}$ a pattern for this mining camp if each sv_i is a pattern component for S.**

An instance of a pattern is a set of records whose values of grouping attributes agree and whose itemizing attributes match those in the pattern.

DEFINITION 3. *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 $r_i, r_j \in R$.*

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).

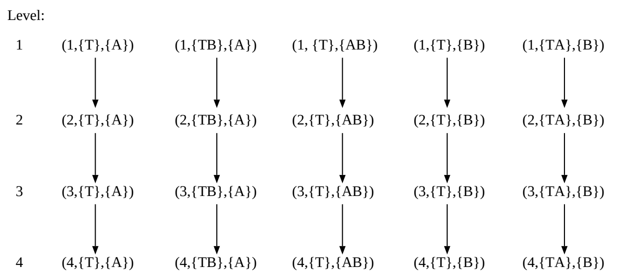

Figure 2: A Simple Search Space

In *FARM*, G and S need not to 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 $S = \emptyset$).

FARM defines a rich set of interrelationships among different mining camps. Consider a mining camp (n, G, S), and attribute $A_i \notin S$. 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 the downward closure property.

Definition 4. *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.*

We use $succ(C_p, C_s)$ to denote C_s is a successor of C_p.

Figure 3 depicts the predecessor and successor relationships present among different mining camps. The root precedes all other mining camps. (In this case, it is not a real camp since $S = \emptyset$.) Now we can define mining spaces.

Definition 5. *A* ***mining space*** *$MS(C)$ is a partially ordered set (poset) of mining camps containing C and all of its successors.*

We can group mining camps to levels.

Definition 6. *The level of mining camp (n, G, S) is defined as $n + |G| + |S|$. We use L_k to denote all mining camps of level k.*

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.

The organization of the mining space in Figure 3 is justified by the downward closure property between each predecessor and successor camps. Exploiting these properties provides considerable benefit in terms of efficiency.

Theorem 1. *Let $C = (n, G, S)$ be a mining camp. Assume the support of pattern $p = \{sv_1, \cdots, sv_n\}$ in C is less than threshold θ, we have:*

1. *For any pattern p' in a type-1 successor of C, and p' is a superset of p, the support of p' is less than θ.*
2. *For any pattern p' in a type-2 successor of C, the support of p' is less than θ.*
3. *For any pattern $p' = \{sv'_1, \cdots, sv'_n\}$ in a type-3 successor of C, and $sv_i \subset sv'_i$ for all $1 \leq i \leq n$, the support of p' is less than θ.*

3. ATTRIBUTE SPECIFICATION LANGUAGE

In this section, we present the Attribute Specification Language (ASL) designed to express domain knowledge for the underlying relational data in mining. The language is small and easy to comprehend, at the same time it allows most types of domain knowledge to be specified easily. It is also a high level specification which hides the inference mechanism behind it. The ASL is essentially a set of predicates that specify the properties of the attributes. An attribute specification is a set of ground atoms (well-formed formulae without connectives).

Assume A, A_1, and A_2 are attribute sets.

- $ignore(A)$ means attribute set A has very little significance in analysis and should be completely left out. For example, attributes with unique values (*Rec* in Figure 1), and other numerical attributes that are not appropriate for frequent itemset mining can be ignored.
- $follow(A_1, A_2)$ specifies those attributes that by themselves are insufficient to form an independent semantic unit, and thus must be combined with other attributes. For example, attribute `City` alone does not provide sufficient information for the location. There are six *Orange* counties/cities and 24 *Springfield* cities in the U.S. To avoid this ambiguity, users can simply specify $follow(\{City\}, \{State\})$. With this, `State` can be used freely but whenever `City` is used, `State` must be used as well.
- $decide(A_1, A_2)$ specifies functional dependency: attribute set A_1 uniquely determines attribute set A_2.
- $together(A)$ means no subset of attribute set A can be used independently. In other words, all attributes in A together forms an atomic semantic unit.
- $item\_only(A)$ specifies that attributes in A, when used, should only be used as itemizing attributes.
- $group\_only(A)$ specifies that attributes in A, when used, should only be used as grouping attributes.
- $always\_group(A)$ means A should always be included as grouping attributes.
- $always\_group(A)$ means A should always be included as grouping attributes.
- $repel\_group(A)$ means no two elements of A can be used as grouping attributes together.
- $repel\_item(A)$ means no two elements of A can be used as itemizing attributes together.
- $repel(A)$ means no two elements in A should appear together in the same mining camp.

We define an **ASL specification**, or simply **specification**, as a set of instantiated atomic formulae using the above predicates.

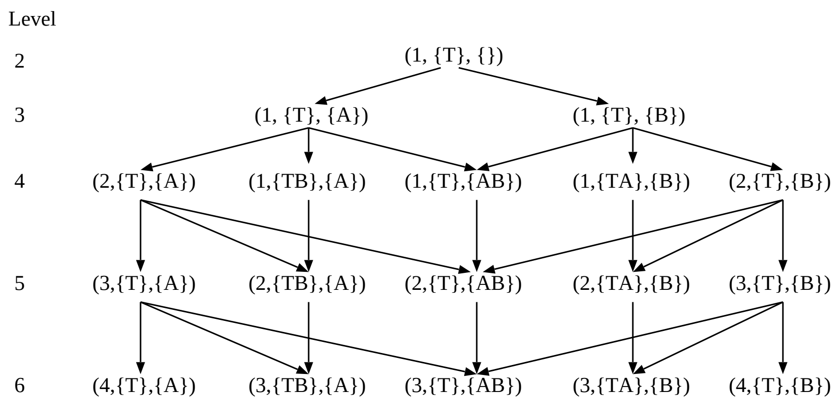

Figure 3: Search Space $MS(1, \{T\}, \{\})$ for attribute set $\{T, A, B\}$

4. THE INFERENCE SYSTEM

In this section, we describe the inference system designed to translate specifications to operational instructions, which in turn provides guidance in mining space exploration. The inference system is a deductive system with a mixture of proof-theoretic and model-theoretic operations. We first discuss how to find the semantic closure of user-specification. Then we describe the rules that identify three types of mining camps that can be eliminated based on the semantic closure. Finally, we present a high level algorithm of the improved *FARM*.

4.1 Semantic Closure of Specifications

The attribute specification language allows users to express domain knowledge in a rather casual style. For example, one user may specify $decide(\{a\}, \{b\})$ and $decide(\{a, b\}, \{c\})$, and another user may specify $decide(\{a\}, \{b\})$ and $decide(\{a\}, \{c\})$ but the two specifications are essentially same. Semantic closure is used here to as a more expression-neutral knowledge base. For a specification, its semantic closure is all the atomic formulae implied by it. The following definition lists the axioms that extend a specification to its semantic closure.

DEFINITION 7. *The ASL semantic closure axiom set, denoted as SC, contains the universally quantified closure of the following formulae.*

1. $decide(X, Y) \longleftarrow (\exists X' \subseteq X, \exists Y' \supseteq Y) decide(X', Y')$.
2. $decide(X_1 \cup X_2, Y_1 \cup Y_2) \longleftarrow decide(X_1, Y_1) \wedge decide(X_2, Y_2)$.
3. $decide(X, Y) \longleftarrow (\exists X')(decide(X, X') \wedge decide(X \cup X', Y))$
4. $follow(X, Y) \longleftarrow (\exists X' \subset X) follow(X', Y)$.
5. $follow(X, Y) \longleftarrow (\exists Y' \supset Y) follow(X, Y')$.
6. $follow(X, Y_1 \cup Y_2) \longleftarrow follow(X, Y_1) \wedge follow(Y_1, Y_2)$.
7. $together(X \cup Y) \longleftarrow together(X) \wedge together(Y) \wedge (X \cap Y \neq \emptyset)$.
8. $ignore(X') \longleftarrow group\_only(X) \wedge item\_only(Y) \wedge X' = X \cap Y$.
9. $ignore(X') \longleftarrow together(X) \wedge repel(Y) \wedge X' = X \cap Y$.

Now we can define the semantic closure of a specification. The symbols used here follow the convention in mathematical logic[6].

DEFINITION 8. *Given SP, a set of atomic formulae in ASL, its semantic closure, SP^*, is defined as*

$$SP^* = \{F | SC \cup SP \vdash F\}$$

The introduction of semantic closure gives more freedom to specification writers in expressing their intention. Here we use the example mentioned earlier to demonstrate different specifications can have the same semantic closure. Let

$$SP_1 = \{decide(\{a\}, \{b\}), decide(\{a, b\}, \{c\})\}$$

and

$$SP_2 = \{decide(\{a\}, \{b\}), decide(\{a\}, \{c\})\},$$

We have

$$SP_1, SC \vdash decide(\{a\}, \{c\})$$

by axiom SC (3), and

$$SP_2, SC \vdash decide(\{a, b\}, \{c\})$$

by axiom SC 1. So we conclude $SP_1 \subset SP_2^*$ and $SP_2 \subset SP_1^*$, hence $SP_1^* = SP_2^*$.
Note the semantic closure is not intended or necessary to be complete in the sense of first order predicate logic. However, this is not a real issue, as it will become clear soon, the coverage of predicates may overlap but it is relatively easy to check whether a mining camp conflicts with a specification.

4.2 Reasoning on Mining Camps

4.2.1 *Mining Camps Classification*

We start by introducing the predicate *nir*. Assume C is a mining camp, $nir(C)$ means mining C yields **n**o **i**nteresting **r**esult. The goal of the inference system is to identify all mining camps that that yields no interesting result. There are three different reasons that $nir(C)$ holds for a mining camp C:

1. The output of C is empty, or $empty(C)$
2. The output of the mining camp is not of users' interest, or $forbiden(C)$.

3. The results of the mining camp C can be inferred by results of another mining camp C', or $reducible(C, C')$.

That is,

$nir(C) \equiv empty(C) \vee forbiden(C) \vee (\exists C')reducible(C, C')$

The rest of this subsection discuss how to check whether a mining camp is empty, forbidden or reducible.

4.2.2 Inferring Empty Camps

There are two ways to validate $empty(C)$. First, given a dataset D, we can check whether there exists any interesting frequent itemset of C in D and we use $D \models empty(C)$ to denote that there is no frequent itemset in C. This is called model-theoretical validation. Second, given a set of formulae F, we can try prove that F implies $empty(C)$ by standard inference rules like modus ponens or resolution[14]; and we use $F \vdash empty(C)$ to denote $empty(C)$ is deducible from F. This is called a proof-theoretical validation.
In *a priori* and original *FARM*, the empty property can only be inferred from validated formulae of the empty predicate. With users' specification, the following lemma also infers empty mining camps.

LEMMA 1. *The axiom set of emptiness, EA, contain the following formula:*

$$empty((n, G, S)) \longleftarrow (n \geq 2) \wedge decide(G, S)$$

We have the following corollary of Theorem 1.

COROLLARY 1. $(\forall C_p, C_s)(empty(C_p) \wedge succ(C_p, C_s) \longrightarrow emtpy(C_s))$.

Model-theoretical and proof-theoretical approaches in reality are very different in cost and effect. Model-theoretical validation requires scanning the data set, an expensive process but Proof-theoretical validation is very simple because the logic system we proposed in this paper is obvious finite, decidable and fast. In fact, we can re-interpret the *a priori* principle[2] as a method to maximize the use of proof-theoretical operations and minimize the use of model-theoretical operations.

4.2.3 Inferring Forbidden Camps

Inferring forbidden camps is fortunately a purely proof-theoretical operation. There are two types of forbidden camps. Type 1 refers to those camps not only themselves are not of interest, all their successors are not of interest too. Type 2 forbidden camps are those themselves are not of interest but their successors maybe of interest.
The following axiom set defines how to validate whether a mining camp is forbidden.

DEFINITION 9. *The axiom set for type 1 forbidden property, $FA1$, includes the following formulae:*

1. $forbidden1((n, G, S)) \longleftarrow (\exists X)(ignore(X) \wedge ((X \cap G \neq \emptyset) \vee (X \cap S \neq \emptyset)))$.
2. $forbidden1(n, G, S) \longleftarrow (\exists X, Y)(follow(X, Y) \wedge (X \subseteq G) \wedge (Y \cap S \neq \emptyset))$.
3. $forbidden1(n, G, S) \longleftarrow (\exists X, Y)(follow(X, Y) \wedge (X \subseteq S) \wedge (Y \cap G \neq \emptyset))$.
4. $forbidden1((n, G, S)) \longleftarrow (\exists X)(together(X) \wedge (X \cap G \neq \emptyset) \wedge (X \cap S \neq \emptyset))$.
5. $forbidden1((n, G, S)) \longleftarrow (\exists X)(group\_only(X) \wedge X \subseteq S)$.
6. $forbidden1((n, G, S)) \longleftarrow (\exists X)(item\_only(X) \wedge X \subseteq G)$.
7. $forbidden1((n, G, S)) \longleftarrow (\exists X)(always\_group(X) \wedge X \cap G \neq \emptyset)$.
8. $forbidden1((n, G, S)) \longleftarrow (\exists X)(always\_item(X) \wedge X \cap S \neq \emptyset)$.
9. $forbidden1((n, G, S)) \longleftarrow (\exists X x_1 x_2)(repel\_group(X) \wedge \{x_1, x_2\} \subseteq G \cap X)$.
10. $forbidden1((n, G, S)) \longleftarrow (\exists X x_1 x_2)(repel\_item(X) \wedge \{x_1, x_2\} \subseteq S \cap X)$.
11. $forbidden1((n, G, S)) \longleftarrow (\exists X x_1 x_2)(repel(X) \wedge \{x_1, x_2\} \subseteq S \cup G)$.

By the definition, the property $forbidden1$ is downward closed and hence we have the following lemma. It is provable by structural induction, here we show the proof of one selected case.

LEMMA 2. $(\forall C_p, C_s)(forbidden1(C_p) \wedge succ(C_p, C_s) \longrightarrow forbidden1(C_s))$.

PROOF. [(2)] Assume for a mining camp $C = (n, G, S)$ and $(\exists X, Y)(follow(X, Y) \wedge (X \subseteq G) \wedge (Y \cap S \neq \emptyset))$, then

1. For type-1 successor $(n + 1, G, S)$, the condition still holds.
2. For type-2 successors $(n, G \cup \{a_i\}, S)$, $X \subseteq G \cup \{a_i\}$ holds, so the condition holds.
3. For type-3 successors $(n, G, S \cup \{a_i\})$, $Y \cup \{a_i\} \cap S \neq \emptyset$ still holds, so does the condition.

□

DEFINITION 10. *The axiom set for type 2 forbidden property, $FA2$, includes the following formulae:*

1. $forbidden2((n, G, S)) \longleftarrow (\exists X)(always\_item(X) \wedge X \not\subseteq S)$.
2. $forbidden2(n, G, S) \longleftarrow (\exists XY)(follow(X, Y) \wedge (X \subseteq G) \wedge (Y \not\subseteq G))$.
3. $forbidden2(n, G, S) \longleftarrow (\exists X, Y)(follow(X, Y) \wedge (X \subseteq S) \wedge (Y \not\subseteq S))$.
4. $forbidden2((n, G, S)) \longleftarrow (\exists X)(together(X) \wedge (X \cap G \neq \emptyset) \wedge (X \not\subseteq G))$.
5. $forbidden2((n, G, S)) \longleftarrow (\exists X)(together(X) \wedge (X \cap S \neq \emptyset) \wedge (X \not\subseteq S))$.
6. $forbidden2((n, G, S)) \longleftarrow (\exists X)(always\_group(X) \wedge X \not\subseteq G)$.
7. $forbidden2((n, G, S)) \longleftarrow (\exists X)(always\_item(X) \wedge X \not\subseteq S)$.

So, the predicate $forbidden$ is defined as

$$forbidden \equiv (forbidden1(C) \vee forbidden2(C)).$$

4.2.4 Inferring Reducible camps

Validating reducible mining camps is also a purely proof-theoretical operation.

Definition 11. *The reducible axiom set, RA, includes the following formulae:*

1. $reducible((n, G, S), (n, G \backslash G', S)) \longleftarrow decide(G \backslash G', G')$.
2. $reducible((n, G, S), (n, G, S \backslash S')) \longleftarrow decide(S \backslash S', S')$.

The **reducible** property is also downward closed.

Lemma 3. $(\forall C_p C_s C_1)(reducible(C_p, C_1) \wedge succ(C_p, C_s) \longrightarrow (\exists C_2)(reducible(C_s, C_2)))$.

Proof. (1) For a mining camp $C = (n, G, S)$, assume $decide(G \backslash G', G')$ holds, so $reducible((n, G, S), (n, G \backslash G', S))$ holds, then,

1. For type-1 successor $(n+1, G, S)$, $reducible((n+1, G, S), (n+1, G \setminus G', S))$ holds.
2. For type-2 successors $(n, G \cup \{a_i\}, S)$, by 1 of SC, $decide(G \cup \{a_i\} \setminus G', G')$ holds, so $reducible((n, G \cup \{a_i\}, S), (n, G \cup \{a_i\} \setminus G', S))$ holds.
3. For type-3 successors $(n, G, S \cup \{a_i\})$, it is obvious that $reducible((n, G, S \cup \{a_i\}), (n, G \setminus G', S \cup \{a_i\}))$ holds.

(2) similar. $\square$

4.2.5 Implementation of the Inference System

For the predicates we discussed, $forbidden2$ is the only one that is not downward closed, so we have to define a predicates for those downward closed predicates.

$$nir^-(C) \equiv empty(C) \vee forbiden1(C) \vee (\exists C')reducible(C, C'))$$

Combining Corollary 1, Lemma 2, and Lemma 3, we conclude the main theorem of this paper.

Theorem 2. *nir^- is downward closed; that is,*

$$(\forall C_p, C_s)(nir^-(C_p) \wedge succ(C_p, C_s) \longrightarrow nir^-(C_s))$$

.

Now we are ready to present the inference procedure of specifications. For a predicate P of ASL and a set of mining camp CS, we use the notation $P(CS)$ to denote $\{P(C)|C \in CS\}$. Given a mining space $MS(C_r)$, a dataset D, and a specification SP , let $TF = SP \cup SC \cup EA \cup FA1 \cup RA$ and k be the level of C_r. Let L_i be the mining camps of level i, NIR_i, the set of avoidable mining camps of level i, can be computed by the following procedure:

$$\begin{aligned}
NIR_k &= \emptyset \\
NIR^-_{i+1} &= \{C \in L_{i+1} | ((\exists C' \in NIR^*_i) succ(C', C)) \text{ or } \\
&\quad\; TF \cup nir^-(NIR^-_{i+1}) \vdash nir^-(C)\} \\
NIR^+_{i+1} &= \{C \in L_{i+1} \setminus NIR^-_{i+1} | FA2 \cup TF \vdash nir(C)\} \\
NIR^*_{i+1} &= NIR^- \cup \{C \in L_{i+1} \setminus NIR^+_{i+1} | D \models empty(C)\} \\
NIR_{i+1} &= NIR^*_{i+1} \cup NIR^+_{i+1}
\end{aligned}$$

where NIR^*_i is the set of camps that themselves and all their successors can be avoided. It contains the part obtained from proof-theoretical validation, NIR^-_i, and the part from model-theoretical validation; NIR^+_i is the set of camps that are avoidable but some of their successors may still of interest.

$ignore(\{Rec\})$
$ignore(\{Time\})$
$together(\{EventType, Maint\})$
$follow(\{Date\}, \{AM/PM\})$
$follow(\{Intrvl5, Intrvl30\}, \{AM/PM\})$
$group\_only(\{Date, Intrvl5, Intrvl30, AM/PM\})$
$item\_only(\{EventType, Source, Subsource, Severity\})$
$decide(\{EventType\}, \{Subsource\})$
$decide(\{Subsource\}, \{Source\})$
$decide(\{EventType\}, \{Severity\})$
$decide(\{Host\}, \{Site\})$
$repel(\{Intrvl5, Intrvl30\})$

Figure 4: Specification System Management Events

A natural way to implement the inference system is to use Prolog or other logic programming language. There are usual concerns with using logic programming language: monotonicity, termination, performance, etc. These concerns are carefully dealt with in this study. First, all the logic sentences used in the inference system are positive definite Horn clauses and conform the fix-point semantics[18; 19] hence the reasoning is monotonic. Second, there is no function symbols in the system, and set operations involved only subsets of the attribute set, hence termination is guaranteed. Third, there is no need to actually find the complete semantic closures of specifications. The depth-first search method, or SLD-resolution, of Prolog can avoid non-necessary computation. Also, the sizes of specifications rarely exceed a thousand formulae, which can be easily handled by existing Prolog interpreters. So performance is not an issue.

The harnessing of Prolog also makes ASL and the inference system easily extensible. One can add predicates to the language along with a set of formulae that define their semantic closure and axioms for proving empty, forbidden, and reducible properties. Then the inference system should be able to interpret those predicates and identify more avoidable mining camps.

5. CASE STUDY

In this section, we demonstrate how the problem size of the example in Section 1 can be greatly reduced. The data set in the example contains 13 attributes. If no constraint is applied, there are $1,586,131 (= 3^{13} - 2^{13})$ possible mappings. Exhausted mining is obvious not a feasible option. However, manually selecting mappings is not a good option either because it is tedious and prone to miss some interesting mappings. With ASL, domain experts can state the following constraints:

1. $ignore(\{Rec\})$ – The value of `Rec` is a unique so it can not be used as a grouping attribute because no two items have same `Rec` value. It is not an itemizing attribute either because it makes every item distinct to the others.
2. $ignore(\{Time\})$ – `Time`, one-second period, is too short to be used to associate events.

3. $together(\{EventType, Maint\})$ – for event types, we always have to consider whether they really indicate actionable situations or simply a transient signal in maintenance operations.

4. $follow(\{Date\}, \{AM/PM\})$ – When `Date` is used as the only temporal attributes in grouping, events observed in 24-hour periods are grouped together. Domain experts believe 24-hour periods are too long and should be refined to 12 hour or less.

5. $follow(\{Intrvl5, Intrvl30\}, \{AM/PM\})$ – The two interval attributes are ambiguous without specifying AM or PM.

6. $group\_only(\{Date, Intrvl5, Intrvl30, AM/PM\})$ – Temporal are only used as grouping attributes.

7. $item\_only(\{EventType, Source, Subsource, Severity\})$ – These four attributes are all information about the nature of the events and are valuable as association rules. However, it is not clear what are the values and meaning to use them as grouping attributes.

8. $decide(\{EventType\}, \{Subsource\})$ – Each event type can only be emitted from a subsource (a specific agent model).

9. $decide(\{Subsource\}, \{Source\})$ – Each subsource is a sub-model of a event agent.

10. $decide(\{EventType\}, \{Severity\})$ – In this dataset, severities (harmless, warning, minor, major, severe, critical and fatal) of an event are assigned based on the event type.

11. $decide(\{Host\}, \{Site\})$ – Host names are unique across domains and each host only resides in one domain.

12. $repel(\{Intrvl5, Intrvl30\})$ – The two interval attributes are derived from `Time` to study associations within different time granularities so they should be separated from each other.

As stated before, domain experts can specify their knowledge in a casual way without thinking too much about the implication. The inference system will find the semantic closure of the specification as true intention of the domain experts. So many different specifications may have the same intention. For example,
$follow(\{Intrvl5, Intrvl30\}, \{AM/PM\})$ can be broken up to two clauses or merge with
$follow(\{Date\}, \{AM/PM\})$ and become
$follow(\{Date, Intrvl5, Intrvl30\}, \{AM/PM\})$. With $group\_only(\{Date, Intrvl5, Intrvl30, AM/PM\})$ already specified, $repel(\{Intrvl5, Intrvl30\})$ has the same effect as $decide(\{Intrvl5, Intrvl30\})$.
The reduction of mining space is shown in Figure 5. The data set has 13 attributes so levels above 14 all have same number of mining camps. The table shows the numbers of mining camps for the cases of exhausting all mining camps, ignoring 2 attributes, ignoring 3 attributes, and applying the specification in Figure 4.
With the help of domain knowledge, the maximal number of mining camps of a level is reduced from 527,345 to 321. The problem becomes solvable and the resulting patterns have better chance to be interpreted and responded to. The inference system can be used alone without actually mining data; it can list all the remaining mining camps in the mining space for user to specify more domain knowledge for further reduction.

6. CONCLUSION

Advances in association rule mining has come the point to deal with common relational data with multiple attributes. However, the enormity of attribute mappings posts a severe challenge to multi-attribute mining on both the computational complexity and usability of the results. However, minimal domain knowledge of the attributes can provide tremendous reduction on the problem size while not losing any interesting patterns.
We presented a multi-attribute data mining framework with the capability of utilizing domain knowledge about attributes to in association rule discovery. The framework includes two parts. The first part is an attribute specification language, ASL, that allows users to specify what and how attributes should be used. ASL consists a set of predicates with simple and comprehensible semantics. The second part is an inference system that is responsible for determining whether a mapping of attributes to itemizing and grouping attributes may produce any interesting results. The system uses a set of axioms to find the semantic closure, the real intention, of the specification. Then it combines the specification, the semantic closure axioms and axioms of empty, forbidden, and reducible properties to infer the properties of mappings. We presented a case study on system management event mining. With domain knowledge, we can transfer the multi-attribute mining problem from being virtually impossible to solve to a reasonable problem size; and the results are more meaningful.
There are several directions for further utilizing domain knowledge in data mining. On the attribute level, the ASL language can be extended with probabilistic information; and the inference system must be able to perform fuzzy or Bayesian reasoning in guiding mining process.

7. REFERENCES

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

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

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

[4] R. Bayardo. Efficiently mining long patterns from database. In *Int. Conf. Management of Data (SIGMOD)*, pages 85–93, 1998.

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

[6] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2nd edition, December 2000.

Level	2	3	4	5	6	7	8	9	10	11	12	13	14
exhaust	12	210	1750	9175	33727	91939	192523	318748	431168	498686	523250	527345	1586131
ignore 2	10	145	985	4135	11947	25177	40417	51892	57002	58025	58025	58025	58025
ignore 3	9	117	705	2595	6501	11793	16365	18660	19171	19171	19171	19171	19171
user-directed	6	32	86	177	269	313	321	321	321	321	321	321	321

Figure 5: Number of Mining Camps of each level

[7] G. Grahne, L. Lakshmanan, X. Wang, and M. Xie. On dual mining: From patterns to circumstances, and back. In *Int. Conf. Data Engineering (ICDE)*, pages 195–204, 2001.

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

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

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

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

[12] B. Padmanabhan and A. Tuzhilin. Unexpectedness as a measure of interestingness in knowledge discovery, 1999.

[13] C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein. Farm: A framework for exploring mining spaces with multiple attributes. In *IEEE Int. Conf. on Data Mining(ICDM)*, 2001.

[14] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[15] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. On Knowledge And Data Engineering*, 8:970–974, 1996.

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

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

[18] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, pages 253–309, 1955.

[19] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, pages 733–742, October 1976.