User-Defined Association Mining

Ke Wang[1] and Yu He[2]

[1] Simon Fraser University
wangk@cs.sfu.ca
http://www.cs.sfu.ca/ wangk
[2] National University of Singapore
hey@comp.nus.edu.sg
http://www.comp.nus.edu.sg/ hey

Abstract. Discovering interesting associations of events is an important data mining task. In many real applications, the notion of association, which defines how events are associated, often depends on the particular application and user requirements. This motivates the need for a general framework that allows the user to specify the notion of association of his/her own choices. In this paper we present such a framework, called the *UDA mining* (*User-Defined Association Mining*). The approach is to define a language for specifying a broad class of associations and yet efficient to be implemented. We show that (1) existing notions of association mining are instances of the UDA mining, and (2) many new ad-hoc association mining tasks can be defined in the UDA mining framework.

1 Introduction

Interesting association patterns could occur in diverse forms. Early work has defined and mined associations of different notions in separate frameworks. For example, association rules are defined by confidence/support and are searched based on the Apriori pruning [1]; correlation rules are defined by the χ^2 statistics test and are searched based on the upward-closed property of correlation [4]; causal relationships are defined and searched by using CCC and CCU rules [14]; emerging patterns are defined by the growth ratio of support [6]. With such an "one-framework-per-notion" paradigm, it is difficult to compare different notions and identify commonalities among them. More importantly, the user may not find such pre-determined frameworks suitable for his/her specific needs. For example, at one time the user likes to find all pairs $< p, c >$ such that p is some above mentioned association pattern and c is a condition under which p occurs; at another time the user likes to know all triples $< p, c_1, c_2 >$ such that association pattern p occurs in the special case c_1 but not in the general case c_2; at yet another time the user wants something else. Even for this simple example, it is not clear how the above existing frameworks can be extended to such "ad-hoc" mining. The topic of this paper is to address this extendibility.

Our approach is to propose a *language* in which the user himself/herself can *define* a new notion of association (vs. *choose* a pre-determined notion).

In spirit, this is similar to database querying in DBMS, in that it does not predict the mining tasks that the user might require to perform; it is the expressive power of the language that determines the class of associations specifiable in this approach. The key is the notion of "user-defined associations" and its specification language. Informally, a user-defined association has two components, *events* and their *relationship*. An event is a conjunction of atomic descriptors, called items, for transactions in the database. For example, event $FEMALE \wedge YOUNG \wedge MANAGER$ is a statement about individuals, where items $FEMALE$, $YOUNG$, and $MANAGER$ are atomic descriptors of individuals [1]. A relationship is a statement about how events are associated. We illustrate the notion of "user-defined associations" through several examples.

Example I. A liberate notion of association between two events X and Y can occur in the form that X *causes* Y. As pointed out by [14], such causal relationships, which state the *nature* of the relationships, cannot be derived from the classic association rules $X \rightarrow Y$. [14] has considered the problem of mining causal relationships among 1-item events, i.e., events containing a single item. In the interrelated world, causal relationships occur more often among multi-item events than among 1-item events. For example, 2-item event $MALE \wedge POSTGRAD$ more likely causes $HIGH\_INCOME$ than each of the 1-item events $MALE$ and $POSTGRAD$ does. Though the concept of causal relationships remains unchanged for multi-item events, the search for such causal relationships turns out to be more challenging because it is unknown in advance which items form a meaningful multi-item event in a causal relationship. In our approach, such general causal relationships are modeled as a special case of user-defined associations.

Example II. The user likes to know all three events Z_1, Z_2, X such that X is more "associated" with Z_1 than with Z_2, where the notion of association between X and Z_i could be any user-defined associations. For example, if $X = HIGH\_INCOME$ is more correlated with $Z_1 = POSTGRAD \wedge MALE$ than with $Z_2 = POSTGRAD \wedge FEMALE$, the user could use it as an evidence of gender discrimination because the same education does not give woman the same pay as man. Again, multi-item events like $POSTGRAD \wedge MALE$ are essential for discovering such associations.

Example III: Sometimes, the user likes to know all combinations of events $Z_1, Z_2, X_1, \ldots, X_k$ such that the association of k events $X_1, \ldots, X_k$, in whatever notion, has sufficiently changed when the condition changes from Z_1 to Z_2. For example, $X_1 = BEER$ and $X_2 = CHIPS$ could be sold together primarily during $Z_1 = [6PM, 9PM] \wedge WEEKDAY$. Here, Z_2 is implicitly taken as $\emptyset$, representing the most general condition.

This list can go on, but several points have emerged and are summarized below.

1. *User-defined associations*. A powerful concept in user-defined association is that the user defines a class of associations by "composing" existing user-

[1] A better term for things like $FEMALE$, $YOUNG$, and $MANAGER$ is perhaps "feature" or "variable". We shall use the term "items" to be consistent with [1].

defined association. The basic building blocks in this specification, such as support, confidence, correlation, conditional correlation, etc., may not be new and, in fact, are well understood. What is new is to provide the user with a mechanism for constructing a new notion of association using such building blocks.

2. *Unified specification and mining.* A friendly system should provide a *single* framework for specifying and mining a broad class of notions of association. We do not expect a single framework to cover all possible notions of association, just as we do not expect SQL to express all possible database queries. What we expect is that the framework is able to cover most important and typical notions of association. We will elaborate on this point in Section 3.
3. *Completeness of answers.* The association mining aims to find *all* associations of a specified notion. In contrast, most work in statistics and machine learning, e.g., model search [18] and Bayesian network learning [7], is primarily concerned with finding some but not all associations. To search for such complete answers, those approaches are too expensive for data sets with thousands of variables (i.e., items) as we consider here.
4. *Unspecified event space.* The event space is not fixed in advance and must be discovered in the search of associations. This feature is different from [3, 4, 14] where only 1-item events are considered. Given thousands of items and that any combination of items is potentially an event, it is a non-trivial task to determine what items make a meaningful event in the association with other events. This task is further compounded by the fact that any combination of events is potentially an association.

In the rest of this paper, we present a unified framework for specifying and mining user-defined associations. The framework must be expressive enough for specifying a broad class of associations. In Section 2 and Section 3, we propose such a framework and examine its expressive power. Equally important, the mining algorithm must have an efficient implementation. We consider the implementation in Section 4. We review related work in Section 5 and conclude the paper in Section 6.

2 User-Defined Association

2.1 Definitions

The database is a collection of *transactions*. Each transaction is represented by a set of Boolean descriptors called *items* that hold on the transaction. An *event* is a conjunction of items, often treated as a set of items. We do not consider disjunction in this paper. $\emptyset$, called the *empty event*, denotes the Boolean constant TRUE or the empty set. Given the transaction database, the *support* of an event X, denoted $P(X)$, is the fraction of the transactions on which event X holds, or of which X is a subset. An event X is *large* if $P(X) \geq mini\_sup$ for the user-specified minimum support $mini\_sup$. Events that are not large occur too infrequently, therefore, do not have statistical significance. The set of large events is downward-closed with respect to the set containment[1]: if X is large and X' is a subset of X, X' is also large. For events X and Y, XY is the shorthand for event $X \wedge Y$ or $X \cup Y$, and $P(X|Y)$ for $P(XY)/P(Y)$. Thus, $X_1, \ldots, X_k$ represents k events whereas $X_1 \ldots X_k$ represents one event $X_1 \wedge \ldots \wedge X_k$. The notion of support can be extended to absence of events. For example, $P(X \neg Y \neg Z)$ denotes the fraction of the transactions on which X holds but neither Y nor Z does.

A user-defined association is written as $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$. $X_1, \ldots, X_k$ are called *subject events*, whose association is of the primary concern. $Z_1, \ldots, Z_p$ are called *context events*, which provide p different conditions for comparing the association of subject events. Context events are always ordered because the order of affecting the association is of interest. The notion of user-defined association $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$ is defined by the support filter and the strength filter defined below.

- $Support\_Filter$. It states that events $X_1, \ldots, X_k, Z_i$ must occur together frequently: if $p > 0$, $P(X_1 \ldots X_k Z_i) \geq mini\_sup$ for $1 \leq i \leq p$; or if $p = 0$, $P(X_1 \ldots X_k) \geq mini\_sup$. In other words, $X_1 \ldots X_k Z_i$, or $X_1 \ldots X_k$ if $p = 0$, is required to be a large event. If this requirement is not satisfied, the co-occurrence of X_i under condition Z_i does not have statistical significance. This condition is called the *support filter* and is written as $Support\_Filter(z_1, \ldots, z_p \rightarrow x_1, \ldots, x_k)$, where z_i and x_i are variables representing the events Z_i and X_i in a user-defined association.
- $Strength\_Filter$: It states that events $Z_1, \ldots, Z_p, X_1, \ldots, X_k$ must hold the relationship specified by a conjunction of one or more formulas of the form $\psi_i \geq mini\_str_i$. Each ψ_i measures some *strength* of the relationship and $mini\_str_i$ is the threshold value on the strength. This conjunction is called the *strength filter* and is written as $Strength\_Filter(z_1, \ldots, z_p \rightarrow x_1, \ldots, x_k)$, where z_i and x_i are variables representing the events Z_i and X_i.

In the above filters, variables x_i and z_i can be instantiated by events X_i and Z_i, and the instantiation is represented by $Support\_Filter(Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k)$ and $Strength\_Filter$ $(Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k)$. Observe that $Support\_Filter(Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k)$ implies that each X_i and Z_i is a large event because of the downward-closed property mentioned earlier. It remains to choose a language for specifying ψ_i, which will determine the class of associations specified and the efficiency of the mining algorithm. We will study this issue shortly. For now, we assume that such a language is chosen. As a convention, we use lower case letters $z_1, \ldots, z_p, x_1, \ldots, x_k$ for event variables and use upper case letters $Z_1, \ldots, Z_p, X_1, \ldots, X_k$ for events.

Definition 1 (The UDA specification). A *user-defined association specification (UDA specification)*, written as $UDA(z_1, \ldots, z_p \rightarrow x_1, \ldots, x_k)$, $k > 0$ and $p \geq 0$, has the form $Strength\_Filter(z_1, \ldots, z_p \rightarrow x_1, \ldots, x_k) \wedge$ $Support\_Filter$ $(z_1, \ldots, z_p \rightarrow x_1, \ldots, x_k)$. (The End)

We say that a UDA specification is *symmetric* if variables x_i's are symmetric in $Strength\_Filter$ (note that variables x_i's are always symmetric in

$Support\_Filter$); otherwise, it is *asymmetric*. A symmetric specification is desirable if the order of subject events does not matter, such as correlation. Otherwise, an asymmetric UDA specification is desirable. For example, an asymmetric UDA is that whenever events $Z_1, \ldots, Z_p$ occur, X_1 occurs but not X_2. We consider only symmetric specification, though the work can be extended to asymmetric specification.

Definition 2 (The UDA problem). Assume that $UDA(z_1, \ldots, z_p \to x_1, \ldots, x_k)$ is given for $0 < k \leq k'$, where $p (\geq 0)$ and $k' (> 0)$ are specified by the user. Consider distinct events $Z_1, \ldots, Z_p, X_1, \ldots, X_k$. We say that $Z_1, \ldots, Z_p \to X_1, \ldots, X_k$ is a *UDA* if the following conditions hold:

1. $X_i \cap X_j = \emptyset$, $i \neq j$, and
2. $X_i \cap Z_j = \emptyset$, $i \neq j$, and
3. $UDA(Z_1, \ldots, Z_p \to X_1, \ldots, X_k)$ is true.

k is called the *size* of the UDA. $Z_1, \ldots, Z_p \to X_1, \ldots, X_k$ is *minimal* if for any proper subset $\{X_{i_1}, \ldots, X_{i_q}\}$ of $\{X_1, \ldots, X_k\}$, $Z_1, \ldots, Z_p \to X_{i_1}, \ldots, X_{i_q}$ is not a UDA. The *UDA problem* is to find all UDAs of the specified sizes $0 < k \leq k'$. The *minimal UDA problem* is to find all minimal UDAs of the specified sizes k. (The End)

Several points about Definition 2 are worth noting.

First, the number of context events, p, in a UDA $Z_1, \ldots, Z_p \to X_1, \ldots, X_k$ is fixed whereas the number of subject events, k, is allowed up to a specified maximum size k_m. This distinction comes from the different roles of these events: for subject events we do not know a prior how many of them may participate in an association, but we often examine a fixed number of conditions for each association. It is possible to allow the number of conditions p up to some maximum number, but we have not found useful applications that require this extension.

Second, context events Z_i's are not necessarily pairwise disjoint. In fact, it is often desirable to examine two context events Z_1 and Z_2 such that Z_1 is a proper superset, thereby a specialization, of Z_2. Then we could specify UDAs $Z_1, Z_2 \to X_1, \ldots, X_k$ such that the association of X_i's holds under the specialized condition Z_1 but not under the general condition Z_2. Other useful syntax constraints could be the requirement on the presence or absence of some specified items in an event, a certain partitioning of the items for context events and subject events, the maximum or minimum number of items in an event, etc. In the same spirit, the disjointness in condition 2 can be removed to express certain overlapping constraints. Constraints have been exploited to prune search space for mining association rules [16, 13]. A natural generalization is to exploit syntax constraints for mining general UDAs. In this paper, however, we focus on the basic form in Definition 2.

2.2 Examples

In this section, we intend to achieve two goals through considering several examples of UDA specification: to show that disparate notions of association can be specified in the UDA framework, and to readily convey the basic idea that underly more complex specification. Once these are understood, the user can define any notion of association of his/her own choice, in the given specification language. We shall focus on specifying $Strength\_Filter$ because specifying $Support\_Filter$ is straightforward. In all examples, lower-case letters z_i and x_i represent event variables and upper-case letters Z_i and X_i represent events.

Example 1 (Association rules). Association rules $Z \rightarrow X$ *introduced in [1] can be specified by*

$$Support\_Filter(z \rightarrow x) : P(zx) \geq mini\_sup$$
$$Strength\_Filter(z \rightarrow x) : \psi(z, x) \geq mini\_conf,$$

where $\psi(z, x) = P(x|z) = P(xz)/P(z)$ *is the* confidence *of rule* $Z \rightarrow X$ *[1]. To factor in both "generality" and "predictivity" of rules in a single measure, the following* $\psi(z, x)$*, called the* J-measure *[15], can be used:*

$$P(z)[P(x|z)log_2\frac{P(x|z)}{P(x)} + (1 - P(x|z))log_2\frac{1 - P(x|z)}{1 - P(x)}].$$

Here, $P(z)$ *weighs the generality of the rule and the term in the square bracket weighs the "discrimination power" of* z *on* x*. This example shows how easy it is to adopt a different definition of association in the UDA framework. (The End)*

In the above specification, the most basic constructs are the supports $P(Z)$, $P(X)$, $P(ZX)$, $P(\neg X)$, $P(Z\neg X)$. Since $Support\_Filter(Z \rightarrow X)$ implies that each of Z, X, ZX is a large event, these supports are readily available from mining large events (note that $P(\neg X) = 1 - P(X)$ and $P(Z\neg X) = P(Z) - P(ZX)$).

Example 2 (Multiway correlation). The notion of correlation is a special case of UDAs without context event. In particular, events $X_1, \ldots, X_k$ *are* correlated *if they occur together more often than expected when they are independent. This notion can be specified by the* χ^2 *statistic test,* $\chi^2(x_1, \ldots, x_k) \geq \chi^2_\alpha$ *[4]. Let* $R = \{x_1, \neg x_1\} \times \ldots \times \{x_k, \neg x_k\}$ *and* $r = r_1 \ldots r_k \in R$*. Let* $E(r) = N * P(r_1) * \ldots * P(r_k)$*, where* N *is the total number of transactions.* $\chi^2(x_1, \ldots, x_k)$ *is defined by:*

$$\Sigma_{r \in R}\frac{(N * P(r) - E(r))^2}{E(r)}.$$

The threshold value χ^2_α *for a user-specified significance level* α*, usually 5%, can be obtained from statistic tables for the* χ^2 *distribution. If* $X_1, \ldots, X_k$ *passes the test,* $X_1, \ldots, X_k$ *are correlated with probability* $1 - \alpha$*. The* uncorrelation *of* $X_1, \ldots, X_k$ *can be specified by* $Strength\_Filter$ *of the form* $1/\chi^2(x_1, \ldots, x_k) \geq 1/\chi^2_\alpha$*, where* α *is usually 95%. If* $X_1, \ldots, X_k$ *passes the filter,* $X_1, \ldots, X_k$ *are uncorrelated with probability* α*. (The End)*

The problem of mining correlation among single-item events was studied in [3,4]. One difference of correlation specified as UDAs is that each event X_i can involve multiple items, rather than a single item. One such example is the correlation of $X_1 = INTERNET$ and $X_2 = YOUNG \wedge MALE$, where $YOUNG \wedge MALE$ is a 2-item event. This generalization is highly desirable because single-item events like $X_1 = INTERNET$ and $X_2 = YOUNG$ or $X_1 = INTERNET$ and $X_2 = MALE$ may not be strongly corrected. It is not clear how the mining algorithms in [3,4] can be extended to multi-item events. A more profound difference, however, is that, as UDAs, we can model "ad-hoc" extension of correlation. The subsequent examples shows this point.

Example 3 (Conditional association). In conditional association $Z \rightarrow X_1, \ldots, X_k$, *subject events* $X_1, \ldots, X_k$ *are associated when conditioned on* Z. *For example,*

$$INT'L \wedge BUSINESS\_TRIP \rightarrow CEO, FIRST\_CLASS$$

says that $X_1 = CEO$ *and* $X_2 = FIRST\_CLASS$ *(flights) are associated for* $Z = INT'L \wedge BUSINESS\_TRIP$ *(international business trips). For example, if the association of* $X_1, \ldots, X_k$ *is taken as the correlation, we have* conditional correlation *defined by* $Strength\_Filter(z \rightarrow x_1, \ldots, x_k)$:

$$\frac{P(x_1 \ldots x_k|z)}{P(x_1|z) * \ldots * P(x_k|z)} \geq mini\_str \tag{1}$$

or alternatively, by the χ^2 *statistic test after replacing* $P(r)$ *and* $P(r_i)$ *in* $\chi^2(X_1, \ldots, X_k) \geq \chi^2_\alpha$ *with* $P(r|z)$ *and* $P(r_i|z)$. *(The End)*

Example 4 (Comparison association). In comparison association $Z_1, Z_2 \rightarrow X$, *subject event* X *is associated* differently *with context events* Z_1 *and* Z_2. *For example,*

$$INT'L \wedge BUSINESS\_TRIP, PRIVATE\_TRIP \rightarrow CEO \wedge FIRST\_CLASS$$

says that $X = CEO \wedge FIRST\_CLASS$ *is more associated with* $Z_1 = INT'L \wedge BUSINESS\_TRIP$ *than with* $Z_2 = PRIVATE\_TRIP$. *To compare two associations for difference, we can compare their corresponding strength* ψ_j *in* $Strength\_Filter$. *In particular, suppose that* $UDA(\rightarrow z_i, x)$ *specifies the association of* Z_i *and* X, $i = 1, 2$. *For each* $\psi_j(z_i, x)$ *in* $UDA(\rightarrow z_i, x)$, $Strength\_Filter(z_1, z_2 \rightarrow x)$ *for the comparison association contains the formula:*

$$Dist(\psi_j(z_1, x), \psi_j(z_2, x)) \geq mini\_str_j. \tag{2}$$

Here, $Dist(s_1, s_2)$ *measures the distance between two strengths* s_1 *and* s_2. *Typical distance measures are* $Dist(s_1, s_2) = s_1/s_2$ *or* $Dist(s_1, s_2) = s_1 - s_2$. *(The End)*

Example 5 (Emerging association). In emerging association $Z_1, Z_2 \rightarrow X_1, \ldots, X_k$, *the association of* $X_1, \ldots, X_k$ *has changed sufficiently when the condition changes from* Z_1 *to* Z_2. *Suppose that* $UDA(z_i \rightarrow x_1, \ldots, x_k)$ *specifies the conditional association* $Z_i \rightarrow X_1, \ldots, X_k$, $i = 1, 2$, *as in Example 3. Then, for each strength function* ψ_j *in* $UDA(z_i \rightarrow x_1, \ldots, x_k)$, $Strength\_Filter(z_1, z_2 \rightarrow x_1, \ldots, x_k)$ *for emerging association contains the formula:*

$$Dist(\psi_j(z_1, x_1, \ldots, x_k), \psi_j(z_2, x_1, \ldots, x_k)) \geq mini\_str_j. \tag{3}$$

The notion of emerging association is useful for identifying trends and changes. For example, the notion of emerging patterns *[6] is a special case of emerging associations of the form $Z_1, Z_2 \to X$, where Z_1 and Z_2 are identifiers for the two originating databases of the transactions. An emerging pattern $Z_1, Z_2 \to X$ says that the ratio of the support of X in the two databases identified by Z_1 and Z_2 is above some specified threshold. To specify emerging patterns, we first merge the two databases into a single database by adding item Z_i to every transaction coming from database i, $i = 1, 2$, and specify $\psi_j(z_i, x) = P(z_i x)/P(z_i)$, where z_i are variables for Z_i, and $Dist(s_1, s_2) = s_1/s_2$ in Equation 3. With the general notion of emerging association, however, we can capture a context Z_i as an arbitrary event (not just a database identifier) and the participation of more than one subject event. (The End)*

In Examples 3, 4 and 5, the "output" UDAs (i.e., conditional association, comparison association, emerging association) are defined in terms of "input" UDAs. These input UDAs are of the form $\to X_1, \ldots, X_k$ in Example 3, $\to Z_i, X$ in Example 4, and $Z_i \to X_1, \ldots, X_k$ in Example 5, which themselves can be defined in terms of their own input UDAs. The output UDAs can be the input UDAs for defining other UDAs. In general, new UDAs are defined by "composing" existing UDAs. It is such a composition that provides the extendibility for defining ad-hoc mining tasks. We further demonstrate this extendibility by specifying causal relationships.

Example 6 (Causal association). Information about statistical correlation and uncorrelation can be used to constrain possible causal relationships. For example, if events A and B are uncorrelated, it is clear that there is no causal relationship between them. Following this line, [14] identified several rules for inferring causal relationships, one of which is the so-called CCC rule: if events Z, X_1, X_2 are pairwise correlated, and if X_1 and X_2 are uncorrelated when conditioned on Z, one of the following causal relationships exists:

$$X_1 \Leftarrow Z \Rightarrow X_2 \qquad X_1 \Rightarrow Z \Rightarrow X_2 \qquad X_1 \Leftarrow Z \Leftarrow X_2,$$

where $\Leftarrow$ means "is caused by" and $\Rightarrow$ means "causes". For a detailed account of this rule, please refer to [14]. We can specify the condition of the CCC rule by $Strength\_Filter(z \to x_1, x_2)$:

$$\begin{array}{c} \psi_1(\emptyset, x_1, x_2) \geq mini\_str_1 \wedge \psi_1(\emptyset, x_1, z) \geq mini\_str_1 \wedge \\ \psi_1(\emptyset, x_2, z) \geq mini\_str_1 \wedge \psi_2(z, x_1, x_2) \geq mini\_str_2. \end{array}$$

Here, $\psi_1(w, u, v) \geq mini\_str_1$ tests the correlation of u and v conditioned on w, and $\psi_2(w, u, v) \geq mini\_str_2$ tests the uncorrelation of u and v conditioned on w. These tests were discussed in Examples 2 and 3. (The End)

In all the above examples, the basic constructs used by the specification are the support of the form $P(v)$, where v is a conjunction of terms $Z_i, \neg Z_i, X_i, \neg X_i$. This syntax of v completely defines the language for $Strength\_Filter$ because we make no restriction on how $P(v)$ should be used in the specification. In the next section, we define the exact syntax for v.

3 The specification language

The term "language" is more concerned with what it can do than how it is presented. There are two considerations in choosing the language for strength functions ψ_i. First, the language should specify a wide class of association. Second, the associations specified should have an efficient mining algorithm. We start with the efficiency consideration. To specify UDAs $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$, we require each strength ψ_i to be above some minimum threshold, where ψ_i is a function of $P(v)$ and v is a conjunction of X_i and Z_j. The support filter $P(X_1 \ldots X_k Z_j) \geq mini\_sup$ is used to constrain the number of candidate Z_j and X_i. The support filter implies that a conjunction v consisting of any number of subject events X_i and zero or one context event Z_j is large. Therefore, supports $P(v)$ for such v are available from mining large events if we keep the support for each large events.

The question is whether it is too restrictive to allow at most one Z_j in each v. It turns out that this is a desirability not a restriction. In fact, each Z_j serves as an individual context for the association of $X_1, \ldots, X_k$ and there is no need to consider more than one Z_j at a time. Another question is that, if absences of events, i.e., $\neg X_i$ and $\neg Z_j$, are desirable in v, as in the examples in Section 2.2, can $P(v)$ be computed efficiently? The next theorem, which is essentially a variation of the well known "inclusion-exclusion" theorem, shows that such $P(v)$ can be computed by the supports involving no absence of events.

Theorem 1. Let $V = \{V_1, \ldots, V_q\}$ be q events of the form X_i or Z_i. Let U be a conjunction of events that do not occur in V. Then

$$P(U \neg V_1 \ldots \neg V_q) = \Sigma_{W \subseteq V} (-1)^{|W|} P(UW),$$

where $|W|$ denotes the number of V_i's in W. (The End)

For example, assume that $V = \{X_2, Z_1\}$ and $U = \{X_1\}$, we have $P(X_1 \neg X_2 \neg Z_1) = P(X_1) - P(X_1 X_2) - P(X_1 Z_1) + P(X_1 X_2 Z_1)$. This rewriting conveys two important points: (1) the right-hand side contains no absence, (2) if $X_1 X_2 Z_1$ is large (as required by the support filter), the right-hand side contains only supports of large events, thus, is computable by mining large events. containing no absence. Based on these observations, we are now ready to define the syntax of v for supports $P(v)$ that appear in a strength function.

Definition 3 (Individual-context assumption). Let Z_j be a context event and X_i be a subject event. v satisfies the *ICA (Individual-Context Assumption)* if v is a conjunction of zero or more terms of the form X_i and $\neg X_i$, and zero or one term of the form Z_j and $\neg Z_j$. A support $P(v)$ satisfies the *ICA* if v satisfies the ICA. A strength function ψ satisfies the *ICA* if it is defined using only supports satisfying the ICA. A strength filter satisfies the *ICA* if it uses only strength functions satisfying the *ICA*. The *ICA-language* consists of all UDAs defined by the support filter in Section 2.1 and the strength filter satisfying the ICA. (The End)

For example, X_1X_2, $\neg X_1X_2$, $X_1X_2Z_1$, and $X_1X_2\neg Z_1$ all satisfy the ICA because each contains at most one term for context events, but $X_1X_2Z_1Z_2$ and $X_1X_2Z_1\neg Z_2$ do not. We like to point out that the ICA is a language on support $P(v)$, not a language on how to use $P(v)$ in defining a strength function ψ. This total freedom on using such $P(v)$ allows the user to define new UDAs by composing existing UDAs in any way he/she wants, a very powerful concept illustrated by the examples in Section 2.2. In fact, one can verify that all the strength functions ψ in Section 2.2 are specified in the ICA-language.

We close this section by making an observation on the "computability" of the ICA-language. The support filter implies that any absence-free v satisfying the ICA is a large event. The rewriting by Theorem 1 preserves the ICA because it only eliminates absences. Consequently, the ICA-language ensures that all allowable supports can be computed from mining large events. This addresses the computational aspect of the language.

4 Implementation

We are interested in a unified implementation for mining UDAs. Given that any combination of items can be an event and any combination of events can be a UDA, it is only feasible to rely on effective pruning strategies to reduce the search space of UDAs. Due to the space limitation, we sketch only the main ideas. Assume that the items in an event are represented in the lexicographical order and that the subject events in a UDA are represented in the lexicographical order (we consider only symmetric UDA specifications). We consider $p > 0$ context events; the case of $p = 0$ is more straightforward. Our strategy is to exploit the constraints specified by $Support\_Filter$ and $Strength\_Filter$ as earlier as possible in the search of UDAs. The first observation is that $Support\_Filter$ implies that all subject events X_i and context events Z_i are large. Thus, as the first step we find all large events, say by applying Apriori [1] or its variants. We assume that the mined large events are stored in a hash-tree [1] or a hash table so that the membership and support of large events can be checked efficiently.

The second step is to construct UDAs using large events. For each UDA $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$, $Support\_Filter$ requires that $X_1 \ldots X_k Z_i$ be large for $1 \leq i \leq p$. Therefore, it suffices to consider only the *k-tuples* of the form $(X_1, \ldots, X_k, Z_i)$, where X_i's are in the lexicographical order and $X_1 \ldots X_k Z_i$ makes a large event for all $1 \leq i \leq p$. We can generate such k-tuples and UDAs of size k in a level-wise manner like Apriori by treating events as items: In the kth iteration, a k-tuple $(X_1, \ldots, X_k, Z_i)$ is generated *only if* $(X_1, \ldots X_{k-1}, X_k)$ and $(X_1, \ldots, X_{k-1}, Z_i)$ were generated in the $(k-1)$th iteration and $X_1 \ldots X_k Z_i$ is large. The largeness of $X_1 \ldots X_k Z_i$ can be checked by looking up the hash-tree or hash table for storing large events. Also, the disjointness of X_k and Z_i, required by $Strength\_Filter$, and the lexicographical ordering of $X_1, \ldots, X_k$, can be checked before generating tuple $(X_1, \ldots, X_k, Z_i)$. After generating all k-tuples in the current iteration, we construct a candidate UDA $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$ using p distinct tuples of the form $(X_1, \ldots, X_k, Z_i)$, $i = 1, \ldots, p$, that share the same prefix $X_1, \ldots, X_k$. Any further syntax constraints on Z_i, as discussed in Section 2.1, can be checked here. A candidate $Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k$ is a UDA if $UDA(Z_1, \ldots, Z_p \rightarrow X_1, \ldots, X_k)$ holds. The above is repeated until some iteration k for which no k-tuple is generated.

For mining minimal UDAs, a straightforward algorithm is to first generate all UDAs and then remove non-minimal UDAs. A more efficient algorithm is finding all minimal UDAs without generating non-minimal UDAs. The strategy is to consider subsets of $\{X_1, \ldots, X_k\}$ for subject events X_i's before considering $\{X_1, \ldots, X_k\}$ itself, and prune all supersets from consideration if any subset is found to be a UDA. Since this is essentially a modification of the above algorithm, we omit the detail in the interest of space.

We have conducted several experiments to mine the classes of UDAs considered in Section 2.2 from the census data set used in [14], which contains 63 items and 126,229 transactions. The result is highly encouraging: it discovers several very interesting associations that cannot be found by existing approaches. For example, some strong causal associations were found among general k-item events, as discussed in Example I, but were not found in [14] because only 1-item events are considered there. This fact re-enforces our claim that the uniform mining approach does not simply unify several existing approaches; it also extends beyond them by allowing the user to define *new* notions of association. We omit the detail of the experiments due to the space limit.

5 Related Work

In [8], a language for specifying several pre-determined rules is considered, but no mechanism is provided for the user to specify new notions of association. In [10], it is suggested to query mined rules through an application programming interface. In [12, 17], some SQL-like languages are adopted for mining association rules. The expressiveness of these approaches is limited by the extended SQL. For example, they cannot specify most of the UDAs in Section 1 and 2. In [11, 9], a generic data mining task is defined as finding all patterns from a given pattern class that satisfy some interestingness filters, but no concrete language is proposed for pattern classes and interestingness filters. Finding causal relationships is studied in [5, 14]. None of these works considers the extendibility where the user can define a new mining task.

6 Conclusion

This paper introduces the notion of user-defined association mining, i.e., the UDA mining, and proposes a specification framework and implementation. The purpose is to move towards a unified data mining where the user can mine a database with the same ease as querying a database. For the proposed approach to work in practice, however, further studies are needed in several areas. Our current work has considered only limited syntax constraints on events, and it is important to exploit broader classes of syntax constraints to reduce the search space. Also, a unified mining algorithm may be inferior to specialized algorithms targeted at specific classes of UDAs. It is important to study various optimization strategies for typical and expensive building blocks of UDAs. In this paper, we have mainly focused on the semantics and "computability" (in no theoretic sense) of the specification language. A user specification interface, especially merged with SQL, is an interesting topic.

References

1. R. Agrawal, T. Imielinski, and A. Swami: Mining Association Rules between Sets of Items in Large Datasets. SIGMOD 1993, 207-216
2. R. Agrawal and R. Srikant: Fast Algorithm for Mining Association Rules. VLDB 1994, 487-499
3. C.C. Aggarwal and P.S. Yu: A New Framework for Itemset Generation. PODS 1998, 18-24
4. S. Brin, R. Motwani, and C. Silverstein: Beyond Market Baskets: Generalizing Association Rules to Correlations. SIGMOD 1997, 265-276
5. G.F. Cooper: A Simple Constraint-based Algorithm for Efficiently Mining Observational Databases for Causal Relationships. Data Mining and Knowledge Discovery, No. 1, 203-224, 1997, Kluwer Academic Publishers
6. G. Dong, J. Li: Efficient Mining of Emerging Patterns: Discovering Trends and Differences. SIGKDD 1999, 43-52
7. D. Heckerman: Bayesian Networks for Data Mining. Data Mining and Knowledge Discovery, Vol. 1, 1997, 79-119
8. J. Han, Y. Fu, W. Wang, K. Koperski, O. Zaiane: DMQL: A Data Mining Query Language for Relational Databases. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, 1996, 27-34
9. T. Imielinski and H. Mannila: A Database Perspective on Knowledge Discovery. Communications of ACM, 39(11), 58-64, 1996
10. T. Imielinski, A. Virmani, and A. Abdulghani: DataMine: Application Programming Interface and Query Language for Database Mining. KDD 1996, 256-261
11. H. Mannila: Methods and Problems in Data Mining. International Conference on Database Theory, 1997, 41-55, Springer-Verlag
12. R. Meo, G. Psaila, S. Ceri: A New SQL-like Operator for Mining Association Rules. VLDB 1996, 122-133
13. R.T. Ng, L. V.S. Lakshmanan, J. Han, A Pang: Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. SIGMOD 1998, 13-24
14. C. Silverstein, S. Brin, R. Motwani, J. Ullman: Scalable Techniques for Mining Causal Structures. VLDB 1998, 594-605
15. P. Smyth, R.M. Goodman: An Information Theoretic Approach to Rule Induction from Databases. IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 4, 301-316, August 1992.
16. R. Srikant, Q. Vu, and R. Agrawal: Mining Association Rules with Item Constraints. KDD 1997, 67-73
17. D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, A. Rosenthal: Query Flocks: a Generalization of Association-Rule Mining. SIGMOD 1998, 1-12
18. T.D. Wickens: Multiway Contingency Tables Analysis for the Social Sciences. Lawrence Brlbaum Associates, 1989.