Identifying Interesting Missing Patterns

Bing Liu, Wynne Hsu, Lai-Fun Mun, and Hing-Yan Lee*

Department of Information Systems and Computer Science
National University of Singapore, Lower Kent Ridge Road,
Singapore 119260, Republic of Singapore
{liub, whsu, munlaifu}@iscs.nus.sg

* Information Technology Institute
11 Science Park Road, Singapore Science Park II,
Singapore 117685, Republic of Singapore
hingyan@iti.gov.sg

Technical Report: TRA8/96

Department of Information Systems and Computer Science

National University of Singapore

Identifying Interesting Missing Patterns

Bing Liu, Wynne Hsu, Lai-Fun Mun, and Hing-Yan Lee*

Department of Information Systems and Computer Science
National University of Singapore, Lower Kent Ridge Road,
Singapore 119260, Republic of Singapore
{liub, whsu, munlaifu}@iscs.nus.sg

* Information Technology Institute
11 Science Park Road, Singapore Science Park II,
Singapore 117685, Republic of Singapore
hingyan@iti.gov.sg

Abstract

One of the important issues in data mining is the subjective "interestingness" problem. It has been shown that in many situations a huge number of patterns can be discovered from a database. Most of these patterns are actually useless or uninteresting to the user. But because of the huge number of patterns, it is difficult for the user to identify those patterns that are of interest to him/her. Past research proposed two main measures of subjective interestingness: unexpectedness and actionability. Both these measures focus on helping the user identify interesting discovered patterns. In this paper, we show that missing patterns (absence of some patterns) are interesting too. An approach has been proposed to identify the interesting missing patterns. The proposed approach is an extension of our previous work. In our previous work, we studied the subjective interestingness problem based on the concept of user's expectations and fuzzy set theory. In that study, the discovered patterns are ranked in different ways according to their unexpectedness to the user. In this paper, we examine the extension to our previous work so as to identify the interesting missing patterns.

1. Introduction

In data mining, techniques are constantly being developed and improved for discovering various types of patterns in databases. While these techniques were shown to be useful in numerous applications, new problems have also emerged. One of the major problems is that, in practice, it is all too easy to discover a huge number of patterns in a database [10, 8, 12]. Most of these patterns are actually redundant, useless, or uninteresting to the user. But due to the sheer size of the patterns, it is very difficult for the user to scan through all the discovered patterns, let alone to comprehend and to identify those patterns that are of interest to him/her. Hence, techniques are needed to rank the discovered patterns according to their degrees of interestingness to the user.

Past research have identified a number of measures of "interestingness" of a discovered pattern. These measures include coverage, strength, support, statistical significance, simplicity, unexpectedness, actionability, etc. [8, 11, 1, 9, 12, etc.]. The first five measures are the so-called objective measures [12]. They can be handled easily with techniques that do not require domain and user knowledge. The last two measures are called the subjective measures [12]. They are defined as follows:

- Unexpectedness: Patterns are interesting if they are unexpected or previously unknown to the user.
- Actionability: Patterns are interesting if the user can do something with them to his/her advantage.

It has been noted [9] that in many applications though objective measures are useful, they are insufficient in determining the interestingness of a discovered pattern. Subjective measures are also needed.

So far, the study of interestingness issues has been focused on the discovered patterns, that is, identifying those discovered patterns that are unexpected and/or actionable [9, 12, 5, 6]. In this paper, we show that there is another class of interesting information that we might be able to deduce. This class of interesting information is the so-called missing patterns (the absence of certain patterns). Let us illustrate what a missing pattern is with an example. A particular service organization keeps a databases of its customers (which consists of *large, medium*, and *small* companies). To improve its services to its customers, the organization performs a data mining session. The outcome of the session consists of a set of discovered patterns of which two are listed below:

Pattern 1. **If** Compy_Size = *large* **Then** Service = *service1*

Pattern 2. **If** Compy_Size = *small* **Then** Service1 = *service1*

Pattern 1 says that if the company size is large it uses the service *service1*. Pattern 2 says that if the company size is small, then it also uses *service1*. Immediately we begin to wonder what happens to those companies whose size is *medium*? Why aren't they using *service1*? The realization of this information may lead this particular service organization to probe into the possibilities of modifying *service1* or of doing more promotion to the *medium* size companies in order to attract them to use the service. In other words, the pattern:

If Compy_Size = *medium* **Then** Service = *service1*

is missing and it is actionable and/or unexpected. Like the discovered patterns, not all missing patterns are interesting. In general, the number of missing patterns are huge (usually much more than the number of discovered patterns) because any attribute-value combinations that are not in the set of discovered patterns will be considered as missing patterns. Clearly, it is not useful to list all the missing patterns to the user. Instead, only the subset of interesting missing patterns should be given. So, how can we identify the interesting missing patterns? We believe our proposed technique provides a partial solution. The proposed technique is an extension of our previous research [6]. In that research, a technique, based on the concept of user expectations (i.e., user's previous knowledge or hypotheses about the database) and fuzzy set theory, is used to identify unexpected patterns. The user is first asked to supply a set of expected patterns. This set of expected patterns is then used in a fuzzy matching algorithm to analyze and rank the discovered patterns. Based on the ranking, we can determine how well a discovered pattern conforms to the user's expectations or how far it is different from the user's expectation. With some extensions to the algorithm, we are able to help the user identify unexpected missing patterns. This extension is the focus of this paper.

2. Problem Definition

As mentioned in the introduction, the number of missing patterns are typically huge (and possibly infinite). We denote the set of possible missing patterns as B. There are only a subset I of them that are of interest to the user, i.e., $I \subseteq B$. Obviously, the subset of interesting missing patterns are subjective in the sense that different people may be interested in different subsets of B. A missing pattern becomes interesting if it is unexpected and/or it is useful to the user. Following the measures of interestingness of the discovered patterns, we say that interesting missing patterns are also of two types: unexpected and actionable missing patterns.

Unexpected missing patterns: these are the patterns that the user expects to find in the database, but were not found, neither are their contradictory patterns.

Actionable missing patterns: these are the patterns that are not found in the database, but their absence made them actionable.

Note that there is an important distinction between a missing pattern and an unexpected pattern. For instance, in the example given above, if the user expects companies of all sizes to use *service1*, and suppose the following patterns are present in the set of discovered patterns:

Pattern 1. **If** Compy_Size = *large* **Then** Service = *service1*

Pattern 2. **If** Compy_Size = *small* **Then** Service1 = *service1*

Pattern 3 **If** Compy_Size = *medium* **Then** Service = Does_not_use(*service1*).

Then even though

MissPattern1 **If** Compy_Size = *medium* **Then** Service = *service1*

is not in the discovered set, we do not say that MissPattern1 is a missing pattern since its contradictory pattern (Pattern 3) is present. In general, a missing pattern arises due to the incompleteness of the databases. In our example, it is not likely that the service company will record which customers do not use their service. This omission means that it is impossible to discover Pattern 3 from the database. As a result, MissPattern1 will be considered a missing pattern.

In the above example, the attribute *Compy_Size* takes discrete values. What happens when an attribute takes on continuous values? The situation is similar. Again, we use an example to illustrate. For example, if the user expects the following pattern to be true:

If A1 $\geq$ 5 **Then** C = Yes, where domain(A1) = [0, 80].

Suppose this expectation is indeed true. However, it may be that when C = *Yes*, *A1* only ranges from 5-10 and 35-80 in the database. We can thus conclude that

If A1 > 10, A1 < 35 **Then** C = Yes

is a missing pattern. As the number of conditions in the expected pattern increases, the discovery of missing patterns gets more and more complicated. The details of the algorithm is presented in Section 3 and 4.

It should be noted that if a missing pattern is what the user expects, then it is not interesting. For instance, suppose the user knows beforehand that *medium* size companies do not use their *service1* due to certain restrictions, then informing the user of the same fact is not interesting, unless the user wishes to confirm his/her knowledge. However, if the user expects the *medium* size companies to use their *service1*, and there is no such pattern in the database, then reporting this missing pattern becomes important and useful.

So the key here is that in order to identify interesting missing patterns, the system must know what the user expects in the first place. This expectation can be specified in the form of expected patterns. In this paper, we focus on a particular type of patterns that takes the following form:

If $P_1, P_2, P_3, \ldots, P_n$ **then** C (or $P_1, P_2, P_3, \ldots, P_n \rightarrow C$)

where "," means "and", and P_i is a conditional proposition of the form:

attr OP value,

where *attr* is an attribute name in the database, and *value* is a possible value for the attribute *attr* and $OP \in \{=, \neq, <, >, \leq, \geq\}$ is the operator. C is the consequent, which has the format of

attr = *value*.

This format is commonly used to represent classification patterns and association patterns.

3. Overview of the Technique

In this section, we present a high-level view of the proposed technique. It consists of two main steps:

Step1. The user is asked to provide a set of patterns E that he/she expects to find in the database D, and also an optional number, T, denoting the expected number of tuples in the database that should satisfy (or support) each $E_j \in E$ (both the conditional and the conclusion parts). The user-expected patterns are regarded as fuzzy patterns. A fuzzy pattern has the same syntax as the original pattern, but its attribute values must be described using some fuzzy linguistic variables [13]. See the definition below.

Step 2. The system goes through the database D (1) to compute the correctness of each expected pattern $E_j \in E$, (2) to count the actual number of tuples that satisfy (or support) E_j, and (3) to discover missing patterns with respect to each E_j by consulting the database tuples.

An important issue here is the representation of the expected patterns. Using fuzzy set theory is a natural choice because human knowledge, hypotheses or intuitive feelings are typically fuzzy. Let us now review the definition of a fuzzy linguistic variable [13].

Definition: A fuzzy linguistic variable is a quintuple $(x, T(x), U, G, \tilde{M})$ in which x is the name of the variable; $T(x)$ is the term set of x; that is, the set of names of linguistic values of x, with each value being a fuzzy variable denoted generally by x and ranging over a universe of discourse U; G is a syntactic rule for generating the name, X, of values of x; and M is a semantic rule for associating with each value X its meaning, $\tilde{M}(X)$ which is a fuzzy subset of U. A particular X is called a *term*.

For example, if *speed* is interpreted as a linguistic variable with $U = [1,140]$, then its term set $T(speed)$ could be

$T(speed) = \{slow, moderate, fast, ...\}$.

$\tilde{M}(X)$ gives a meaning to each term. For example, $\tilde{M}(slow)$ may be defined as follows:

$$\tilde{M}(slow) = \{(u, \mu_{slow}(u)) \mid u \in [1, 140]\}$$

$$\text{where } \mu_{slow}(u) = \begin{cases} 1 & u \in [1,30] \\ -\frac{1}{30}u + 2 & u \in (30,60] \\ 0 & u \in (60,140] \end{cases}$$

$\mu_{slow}(u)$ denotes the degree of membership of u in term *slow*.

This may look complicated. However, the user only needs to input the fuzzy set $\tilde{M}(X)$ for each term X used in the expected patterns. A graphical user interface has been implemented to simplify this input process.

4. Detailed Computation in the Proposed Technique

After discussing the basic idea of the proposed technique, we now present the detailed computations in Step 2.

4.1 Computing the correctness of expected pattern E_j

Naturally, when the user provides an expected pattern $E_j \in E$, he/she wishes to know whether the pattern is correct or not. To obtain this correctness value, the system needs to check against the database to see how true E_j is. Since E_j is regarded as a fuzzy pattern, matching E_j with each tuple in the database is also fuzzy. It returns a value in the range [0, 1]. Thus, to consider a match to be satisfactory, a cutoff value needs to be used.

Let *cutoff* be a value in the range (0, 1] denoting the minimal degree that a data tuple $D_k \in D$ must match (or satisfy) the conditional or conclusion part of $E_j \in E$. The value of *cutoff* is specified by the user. Let $|A_j|$ be the number of attributes mentioned in the conditional part of E_j. We denote $V_{a,k,j}$ as the match value of the ath conditional proposition of E_j with D_k, and $Z_{k,j}$ as the match value of the conclusion of E_j with D_k. We then define

$$M_{cond}^{k,j} = \min(V_{1,k,j}, V_{2,k,j}, \ldots, V_{|Aj|,k,j})$$

to be the degree of condition match of D_k and E_j (or the degree that D_k satisfies the conditions of E_j) and similarly, we define $M_{concl}^{k,j}$ (= $Z_{k,j}$) to be the degree of match of the conclusion. The detailed computation of $V_{a,k,j}$ and $Z_{k,j}$ is discussed below. The **correctness** (or **accuracy**) of E_j, $Corr_j$, is hence computed as follows:

$$Corr_j = \frac{\text{Total number of tuples with } M_{cond}^{k,j}, M_{concl}^{k,j} \geq cutoff}{\text{Total number of tuples with } M_{cond}^{k,j} \geq cutoff}$$

4.2 Computing $V_{a,k,j}$ and $Z_{k,j}$

To compute $V_{a,k,j}$ and $Z_{k,j}$, we need to consider both the attribute value and the operator in E_j. In addition, the attribute value types (discrete or continuous) are also important. Since the computations of $V_{a,k,j}$ and $Z_{k,j}$ are the same, it suffices to just consider $V_{a,k,j}$.

4.2.1. Matching of discrete attribute values

In this case, the semantic rule for each term (X) used in describing the expected pattern is defined over the universe (or domain) of the discrete attribute. We denote U as the set of possible values for the attribute. For each $u \in U$, the user needs to input the membership value of u in X, denoted as $\mu_X(u)$. For example, the user gives the following pattern:

If *Grade* = *poor* **then** *Class* = *reject*

Here, *poor* is a fuzzy term. To describe this term, the user needs to specify the semantic meaning for *poor*. Assume the universe of the discrete attribute *Grade* = {*A, B, C, D, F*}. The user may specify that *poor* grade means:

$\{(F, 1), (D, 0.8), (C, 0.2)\}$,

where the left coordinate is an element in the universe of *Grade*, and the right coordinate is the degree of membership of that element in the fuzzy set *poor*, e.g., $\mu_{poor}(D) = 0.8$. It is assumed that all the other attribute values not mentioned in the set have the degree of membership 0. The fuzzy set for *reject* can also be similarly specified by the user.

As mentioned, when evaluating $V_{a,k,j}$, both the fuzzy set of the attribute value description and the operator used play an important role. In the discrete case, the valid operators are "=" and "≠". Suppose the conditional proposition to be matched in E_j is:

attr Opu X

where *attr* is an attribute name, *Opu* belongs to the set {=, ≠}, and *X* is the term. Assume *S* is the value of the attribute *attr* in the data tuple D_k. Two cases result:

Case 1. *Opu* = "=": $V_{a,k,j} = \mu_X(S)$.

Case 2. *Opu* = "≠": $V_{a,k,j} = \mu_{\neg X}(S)$.

4.2.2. Matching of continuous attribute values

When an attribute takes continuous values, the semantic rule for the term (*X*) takes on the form of a continuous function. To simplify the user's task of specifying the shape of this continuous function, we assume the function has a curve of the form as shown in Figure 1. Thus, the user merely needs to provide the values for *a*, *b*, *c*, and *d*.

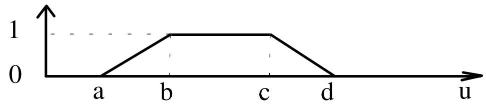

Figure 1. Membership function

For example, the user gives the pattern: **If** *Age* = *young* **then** *Class* = *accept*.

Here, *young* is a term for the variable *Age* whose domain is [0, 80]. To describe the semantic meaning of *young*, the user just need to supply the values of a, b, c, and d. For example, *young* may be described with $a = 15$, $b = 20$, $c = 30$, and $d = 35$.

In the continuous case, the range of values that the operator can take is expanded to {=, ≠, >, <, ≥, ≤, ≤≤, <<}. "≤≤" is used to represent a range like this: $X_1 \leq attr \leq X_2$, and "<<" is used to represent $X_1 < attr < X_2$. With this expansion, the total number of possible cases to be considered is 8. Detailed discussion of the computations for the eight cases can be found in [6].

4.3 Identifying unexpected missing patterns

After computing the correctness of the expected pattern E_j, we now present the technique for identifying unexpected missing patterns. Since our proposed technique is an extension of the technique in [6], let us first give an overview of the technique in [6] for finding unexpected and other types of interesting patterns through post-analysis. The proposed approach for finding unexpected missing patterns follows naturally from the technique in [6].

Table 1 illustrates the example. Assume the database has 12 data tuples (shown in the table), and 10 attributes, i.e., A_1, A_2, ... A_{10} (not shown in the table). Four classification patterns (we use classification patterns here for illustration) are generated from the database, e.g., using C4.5 [11]. We denote *B* as the set of discovered patterns. Let the expected pattern E_j be $PE_j \rightarrow CE_j$, where PE_j is the set of conditions (e.g., $A_1 = 3$, $A_4 = 5$) and CE_j is the conclusion (e.g., $C = Yes$). Let the *i*th discovered pattern, B_i, be $X_i \rightarrow Y_i$, where X_i is the set of conditions (e.g., $A_2 = 3$, $A_4 = 5$) and Y_i is the conclusion (e.g., $C = No$). The table is explained as follows:

- Each row in the table symbolizes a data tuple numbered 1, 2, ..., 12. The actual tuple with all attribute values is not shown.
- There are three main columns in the table. Column 1 shows the tuple numbers. Column 2 shows the expected pattern E_j, $PE_j \rightarrow CE_j$. This column is divided into two sub-columns, one for PE_j and one for CE_j. If a data tuple satisfies PE_j (i.e., $M^{k,j}_{cond} \geq cutoff$) or CE_j (i.e., $M^{k,j}_{concl} \geq cutoff$), then a ✓ appears, otherwise a × appears. Column 3 consists of 4 sub-columns, representing 4 generated patterns. A ✓ in a row means that the data tuple

satisfies the pattern (both the conditional and the conclusion parts) represented by the sub-column. We also say that the tuple is *tagged* under this generated pattern.

- For each tuple, there could be more than one sub-column that has a ✓ because a tuple could be successfully classified by more than one discovered pattern.
- There are some tuples that are not tagged under any B_i (e.g., tuple 4 and 12). This reflects the situation that there exist tuples that do not satisfy any discovered pattern.

For ease of understanding and without loss of generality, we order the tuples in Table 1 according to their satisfaction of PE_j and CE_j. Note that each expected pattern E_j is considered separately.

	Expected pattern $PE_j \rightarrow CE_j$		**Generated patterns and the data tuples they tag**				
Tuple No.	PE_j	CE_j	$X_1 \rightarrow Y_1$	$X_2 \rightarrow Y_2$	$X_3 \rightarrow Y_3$	$X_4 \rightarrow Y_4$	
1	✓	✓	✓				
2	✓	✓	✓				Group 1
3	✓	✓	✓	✓			
4	✓	✓					
5	✓	✗			✓		
6	✓	✗			✓		Group 2
7	✓	✗			✓		
8	✗	✓		✓			
9	✗	✓		✓			
10	✗	✗			✓	✓	Group 3
11	✗	✗				✓	
12	✗	✗					

Table 1 An example for illustration

From the table, we see that $PE_j \rightarrow CE_j$ is not accurate in general because only 4 out of 7 tuples that satisfy PE_j also satisfy CE_j. To further analyze the table, we divide the tuples in the database into 3 groups.

Group 1 (tuple 1-4): This group consists of all the tuples that satisfy both PE_j and CE_j. Within this group, tuples are tagged either under Pattern 1 or Pattern 2. Thus, we say that Pattern 1 and Pattern 2 ***conform*** to $PE_j \rightarrow CE_j$ to certain degree. Among the 4 tuples in this group, 3 of them are tagged under Pattern 1, while only 1 is tagged under Pattern 2. Pattern 1 and Pattern 2 are shown to be conforming to $PE_j \rightarrow CE_j$. They are also ranked according to the number x/y, where x is the number of tuples that are tagged under the pattern within the group, while y is the number of tuples that are tagged under the pattern in the whole database.

Group 2 (tuple 5-7): This group consists of tuples that satisfy PE_j but not CE_j. Within this group, tuples are tagged under one pattern (Pattern 3). Thus, we say that Pattern 3 is ***unexpected*** with respect to $PE_j \rightarrow CE_j$ to certain degree.

Group 3 (tuple 8-12): In this last group, the tuples do not satisfy PE_j and they are tagged under three generated patterns (Pattern 2, 3 and 4). We say that Pattern 2, 3 and 4 are ***unrelated*** to $PE_j \rightarrow CE_j$ to some degree. This group can be further divided into two sub-groups. One sub-group satisfies CE_j (tuple 8 and 9), and the other does not (tuples 10-12).

More details on this technique can be found in [6]. Let us now turn to missing patterns. By definition, a missing pattern is one that the user expects to find but is missing. In other words, a missing pattern is one that should cover some tuples in Group 1, but those tuples are absent in the database. The algorithm for identifying such missing patterns is presented below.

1 Initialize $PatternNo_j$, $CondNo_j$ to 0 for $1 \le j \le |E|$;
2 **For** each expected pattern, E_j **Do**
3   **For** each condition *$attr_i$ op value* in E_j, **Do**
4   Divide the value range of the condition into a number of small buckets
5   **Endfor**
6 **Endfor;**
7 **For** each tuple, D_k, in the database **Do**
8   **For** each expected pattern, E_j, of the form $PE_j \rightarrow CE_j$ **Do**
9   Compute $M_{cond}^{k,j}$, and $M_{concl}^{k,j}$;
10   **If** $M_{cond}^{k,j} \ge cutoff$ **Then**
11   Increment $CondNo_j$; /* number of tuples that satisfy PE_j */
12   **If** $M_{concl}^{k,j} \ge cutoff$ **Then**
13   Increment $PatternNo_j$; /* number of tuples that satisfy both PE_j and CE_j */
14   **For** each condition *$attr_i$ op value* **Do**
15   Increment the $count_{i,j,l}$ of bucket l where the value of $attr_i$ in the tuple D_k falls in;
16   **Endfor;**
17   **Endif**;
18   **Endif**;
19   **Endfor**;
20 **Endfor**;
21 **For** each expected pattern E_j **Do** $Corr_j = \frac{PatternNo_j}{CondNo_j}$; **Endfor**;
22 **For** each expected pattern, E_j **Do**
23   **For** each $attr_i$ in the conditional part of E_j **Do**
24   Group the buckets for $attr_i$ into value ranges according to whether they are filled or not;
25   **For** each *$attr_i$'s* value range R **Do**
26   **If** R is a unfilled range **Then** output a missing pattern **Endif**;
27   **Endfor**;
28   **Endfor;**
29 **Endfor**

We use an example to illustrate how to the algorithm works. Assume an expected pattern E_j is

If A1 > 3, A2 > 7 **Then** C = Yes.

Note that we do not use fuzzy terms for the value 3 and 7 for easy explanation (a crisp pattern is just a special case of a fuzzy pattern). Assume the maximal value of A1 is 13 and the maximal value of A2 is 17.

- The algorithm consists of 4 main steps:

Step 1 (line 2-6): Divide the value range of each condition of E_j into a number of small buckets. For discrete value attribute, each relevant value forms a bucket. In our example, the value ranges of A1 > 3 and A2 > 7 are divided into many small equal size buckets as shown in Figure 2. In our implementation, we set the default number of buckets to 20 (or less, if the value range is small). The user is also allowed to specify a different number of buckets.

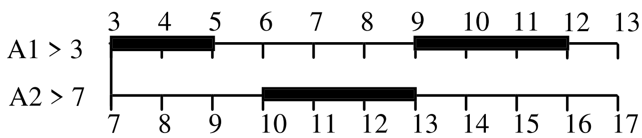

Figure 2. An example

Step 2 (line 7-20): Count the number of tuples that fall in each bucket for each conditional attribute of E_j. Note that each bucket is represented as a counter, $count_{i,j,l}$, which is initialized to 0 at the beginning.

Step 3 (line 21): Compute the correctness of E_j.

Step 4 (line 22-29): Group the buckets for each $attr_i$ in the conditional part of E_j into ranges of two types, *filled* and *unfilled* (line 24). A *filled* range is made up of a number of consecutive filled buckets. A bucket is considered *filled* if it satisfies the following formula:

$$count_{i,j,l} \geq e \times \frac{EN}{\text{The total number of buckets}}$$

where EN is T if the user provides the expected number of tuples, otherwise it is the total number of tuples in the databse that satisfy (or support) E_j, and e is the *cutoff* coefficient (the default value is 0.3). Those buckets that do not satisfy the condition are said to be *unfilled*. In our example, *A1* is grouped into four value ranges, 3-5, 5-9, 9-12, and 12-13 with the dark lines representing the *filled* ranges (see Figure 2). For each *unfilled* range (line 26), we generate a missing pattern of the form:

If <unfilled_range>, <other_conditions_of_ P_{E_j} > **Then** C_{E_j}

Thus, in our example, the following missing patterns will be produced.

If $5 < A1 \leq 9$, $A2 > 7$ **Then** C = Yes

If $A1 > 3$, $7 < A2 \leq 10$ **Then** C = Yes

If $A1 > 3$, $13 < A2 \leq 17$ **Then** C = Yes

Note that there is no missing pattern for A1 > 12 because we assume that a *unfilled* range must be formed by more than two consecutive buckets before a missing pattern will be generated.

At this point, we note that our proposed method for generating missing patterns is, in fact, not complete. For instance, instead of merely generating the missing patterns corresponding to each *unfilled* range, we can record the mapping relationships between the buckets of *A1* and the buckets of *A2* (see Figure 3). Then, more accurate description of the missing patterns can be generated. For example, the following missing pattern can be generated:

If $3 < A1 \leq 5$, ($7 < A2 \leq 10$ or $12 < A2 \leq 17$) **Then** C = Yes

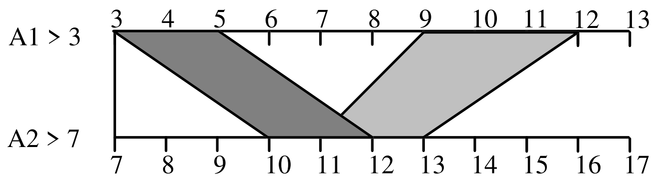

Figure 3. A more complex situation

The drawback of this later approach is that: (1) it can potentially produce a large number of complex missing patterns that serves little useful purpose except to confuse the user, (2) the data structure required to store the mapping information increases exponentially with the number of conditions in E_j. In view of these two drawbacks, we decide to implement the first approach with the objective of alerting the user without overwhelming him/her. Our system will be used as a starting point to identify interesting missing patterns. To further analyze such missing patterns, data visualization tools such as WinViz [4] can be used.

- *$PatternNo_j$* also serves to inform the user whether the number of tuples that satisfy E_j exceeds the expected number *T*. If the number is smaller than *T*, then we can conclude that E_j itself is, in fact, a missing pattern
- Though we strive to make a distinction between a missing pattern and an unexpected pattern in Section 2, for practical implementation, we find that such distinction is difficult to achieve without domain knowledge. For example, suppose the user expects that companies of all sizes use their *service1*. The set of discovered patterns are:

  Pattern 1. **If** Compy_Size = *large* **Then** Service = *service1*
  Pattern 2. **If** Compy_Size = *small* **Then** Service = *service1*
  Pattern 3. **If** Compy_Size = *medium* **Then** Service = *service2*

  Then the pattern:

  If Compy_Size = *medium* **Then** Service = *service1.*

  is unexpected if *service1* and *service2* are mutually exclusive, and hence it should not be reported as a missing pattern. If, however, there is no relationship between *service1* and *service2*, then the pattern will be reported as a missing pattern. In other words, to correctly recognized a pattern as a missing pattern, the system needs to know whether a value has some correlation with other values. This involves asking the user to supply the necessary information. To simplify the task for the user, we have decided to report a pattern as missing pattern regardless of whether an unexpected pattern exists or not. This is reasonable because it only gives a bit of redundant information, and it does not create any confusion.
- Complexity of this algorithm can be analyzed as follows: the computation complexity is dominated by Step 2 and Step 4. Step 1 and 3 requires little computation time. Let the maximal number of conditions of an expected pattern be *N*, the maximal number of buckets for an attribute be *L*, the number of expected patterns be $|E|$ and the database size be $|D|$. Line 9 takes O(*N*) time. Line 15 takes O(1) time. Thus, the time complexity of Step 2 is $O(|E||D|N)$. The time complexity of Step 4 is $O(|E|NSL)$.

5. An Illustration

We use the credit screening database created by Chiharu Sano in UCI repository to illustrate the discovery of unexpected missing patterns. The user-expected patterns are given below:

- **User-expected patterns**

Expected pattern 1:	IF	YR_Work >= Few	{a=2, b=3, c=4, d=5}
	THEN	Granted = YES	{(Yes, 1), (No, 0)}
Expected pattern 2:	IF	Bought = Any	{(bike, 1), (car, 1), (furniture, 1), (jewel, 1), (medinstru, 1), (pc, 1), (stereo, 1)}
		Jobless = NO	{(No, 1), (Yes, 0)}
	THEN	Granted = YES	{(Yes, 1), (No, 0)}

No expected number of tuples that satisfy (or support) either expected patterns is specified by the user.

The running results are reported below (0.6 is used as the *cutoff* match value):

- **Correctness (or accuracy) of the expected patterns**

Expected Pattern 1	Accuracy:	90.0%	Coverage Of Condition : 70 cases Coverage Of Pattern : 63 cases
Expected Pattern 2	Accuracy:	74.8%	Coverage Of Condition : 111 cases Coverage Of Pattern : 83 cases

From the indicated accuracy (or correctness), we can see that the two patterns are quite accurate, which means that the user’s knowledge is quite correct.

- **Missing patterns**

  Missing Patterns for expected pattern 1

  Missing Pattern 1: If 21 < YR_Work <= 24 Then Granted = YES
  System's Estimated Expected Support = 6.3 tuples
  Observed Support = 0 tuples

  Missing Pattern 2: If 30 < YR_Work <= 33 Then Granted = YES
  System's Estimated Expected Support = 6.3 tuples
  Observed Support = 0 tuples

  From these two missing patterns we see that no people whose year of work (which is a continuous attribute) is in the range (21, 24] or (30, 33] was granted a loan. This is rather unexpected and the user may be very interested to know why that is the case.

  Missing Patterns for expected pattern 2

  Missing Pattern 1: If Bought = bike, Jobless = NO Then Granted = YES
  System's Estimated Expected Support = 11.9 tuples
  Observed Support = 3 tuples

  This missing pattern also shows something unexpected because the user expects that as long as the person has a job and bought something, he/she should be granted a loan. However, this is not the situation. In fact, the missing pattern indicates that the database contain very few tuples of a person who has a job, and bought a bike, and was granted a loan.

From this example, we see that the absence of certain patterns can serve to alert the user and prompt the user to perform some actions so as to improve the missing pattern situation.

6. Related Work

Subjective interestingness has long been identified as an important problem in data mining. Current research on subjective interestingness focuses on finding interesting discovered pattern, i.e., unexpected discovered patterns and actionable discovered patterns. To the best of our knowledge, there is no reported work on helping user identify interesting missing patterns.

[9] studied the issue of subjective interestingness in the context of a health care application. The system (called KEFIR) analyzes the health care information to uncover interesting deviations (from the norms) or “key findings”. A domain expert system is built to identify findings that are actionable and the actions to be taken. KEFIR does not identify interesting missing patterns. The KEFIR approach is also application specific. Its domain knowledge is hard-coded in the system as production rules. Our system is domain independent.

[12] proposes to use probabilistic beliefs and belief revision as the framework for describing subjective interestingness. Specifically, a belief system is used for defining unexpectedness. However, [12] is just a proposal, and no system was implemented. This work also does deal with missing patterns. In addition, our approach allows the user to specify his/her beliefs (expectations) in fuzzy terms which we feel are more natural and intuitive than complex conditional probabilities that the user has to assign in [12].

In [5], [6] and [7], we reported two fuzzy matching techniques to perform the post-analysis of patterns in order to help the user to identify unexpected patterns. This work attempts to extend the method in [6] such that interesting missing patterns may also be identified.

7. Conclusion

This paper points out that in some situations, missing patterns may be interesting to the user. A technique is proposed to help the user identifies unexpected missing patterns. This method is characterized by asking the user to provide his/her expectation or existing knowledge, and then the system uses a simple algorithm to find those unexpected missing patterns. The proposed technique is domain and application independent. In our future work, we plan to integrate it with a data visualization system so that all the missing patterns can be displayed graphically to make them easier to understand..

Acknowledgment

We thank Gui-Jun Yang for implementing the graphic user interface and part of the system. This project is funded by Singapore National Science and Technology Board.

References

[1] W. J. Frawely, G. Piatesky-Shapiro, C. J. Matheus. Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro and W.J Frawley (eds), *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991, 1-27.

[2] J. Han, Y. Cai, and N. Cercone. Data driven discovery of quantitative rules in relational database. *IEEE Trans. Knowl. & Data Eng.* 5, 1993, 29-40.

[3] W. Klosgen. Problems for knowledge discovery in databases and their treatment in the statistics interpreter explora. *International Journal of Intelligent Systems*, 7(7), 1992, 649-673.

[4] H.Y Lee, H.L Ong and L.H Quek, Exploiting visualization in knowledge discovery *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995.

[5] B. Liu and W. Hsu. Post-analysis of learned rules. To appear in *AAAI-96*, Aug 4-8, 1996, Portland, Oregon, USA.

[6] B. Liu, W. Hsu, H. Y Lee and L.F. Mun. Tuple-level analysis of learned rules. *Submitted* (also appeared as a *DISCS Technical Report*, National University of Singapore), 1996.

[7] B. Liu, W. Hsu, L.F. Mun and H. Y Lee. Finding interesting patterns using user expectations. *Technical Report TRA7/96*, Department of Information Systems and Computer Science, National University of Singapore, 1996.

[8] J. A. Major, J. J. Mangano. Selecting among rules induced from a hurricane database. *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, 1993, 30-31.

[9] G. Piatesky-Shapiro and C. J Matheus. The interestingness of deviations. *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, 1994, 25-36.

[10] G. Piatetsky-Shapiro, C. Matheus, P. Smyth, and R. Uthurusamy. KDD-93: progress and challenges ... *AI magazine*, Fall, 1994, pg. 77-87.

[11] J. Ross Quinlan. *C4.5*: *program for machine learning*. Morgan Kaufmann, 1992.

[12] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995, 275-281.

[13] H. -J. Zimmermann. *Fuzzy set theory and its applications*. Kluwer Academic Publishers, 1991.