Mining Frequent Itemsets from Uncertain Data

Chun-Kit Chui[1], Ben Kao[1], and Edward Hung[2]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
{ckchui,kao}@cs.hku.hk
[2] Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong
csehung@comp.polyu.edu.hk

Abstract. We study the problem of mining frequent itemsets from *uncertain data* under a *probabilistic framework*. We consider transactions whose items are associated with existential probabilities and give a formal definition of frequent patterns under such an uncertain data model. We show that traditional algorithms for mining frequent itemsets are either inapplicable or computationally inefficient under such a model. A *data trimming* framework is proposed to improve mining efficiency. Through extensive experiments, we show that the data trimming technique can achieve significant savings in both CPU cost and I/O cost.

1 Introduction

Association analysis is one of the most important data-mining model. As an example, in market-basket analysis, a dataset consists of a number of tuples, each contains the items that a customer has purchased in a transaction. The dataset is analyzed to discover associations among different items. An important step in the mining process is the extraction of frequent itemsets, or sets of items that co-occur in a major fraction of the transactions. Besides market-basket analysis, frequent itemsets mining is also a core component in other variations of association analysis, such as association-rule mining [1] and sequential-pattern mining [2].

All previous studies on association analysis assume a data model under which *transactions* capture doubtless facts about the items that are contained in each transaction. In many applications, however, the existence of an item in a transaction is best captured by a likelihood measure or a probability. As an example, a medical dataset may contain a table of patient records (tuples), each of which contains a set of symptoms and/or illnesses that a patient suffers (items). Applying association analysis on such a dataset allows us to discover any potential correlations among the symptoms and illnesses. In many cases, symptoms, being subjective observations, would best be represented by probabilities that indicate

This research is supported by Hong Kong Research Grants Council Grant HKU 7134/06E.

Z.-H. Zhou, H. Li, and Q. Yang (Eds.): PAKDD 2007, LNAI 4426, pp. 47–58, 2007.

Table 1. A diagnosis dataset

Patient ID	Depression	Eating Disorder
1	90%	80%
2	40%	70%

their presence in the patients' tuples. Table 1 shows an example patient dataset. A probability value in such a dataset might be obtained by a personal assessment conducted by a physician, or it could be derived based on historical data statistics. (For example, a patient who shows positive reaction to Test A has a 70% probability of suffering from illness B.) Another example of uncertain datasets is pattern recognition applications. Given a satellite picture, image processing techniques can be applied to extract features that indicate the presence or absence of certain target objects (such as bunkers). Due to noises and limited resolution, the presence of a feature in a spatial area is often uncertain and expressed as a probability [3]. Here, we can model a spatial region as an object, and the features (that have non-zero probabilities of being present in a region) as the items of that object. The dataset can thus be considered as a collection of tuples/transactions, each contains a set of items (features) that are associated with the probabilities of being present. Applying association analysis on such a dataset allows us to identify closely-related features. Such knowledge is very useful in pattern classification [4] and image texture analysis [5].

In this paper we consider datasets that are collections of transactional records. Each record contains a set of items that are associated with *existential* probabilities. As we have mentioned, a core step in many association analysis techniques is the extraction of frequent itemsets. An itemset is considered *frequent* if it appears in a large-enough portion of the dataset. The occurrence frequency is often expressed in terms of a support count. For datasets that contain uncertain items, however, the definition of support needs to be redefined. As we will discuss later, due to the probabilistic nature of the datasets, the occurrence frequency of an itemset should be captured by an *expected* support instead of a traditional support count. We will explain the *Possible Worlds* interpretation of an uncertain dataset [6] and we will discuss how expected supports can be computed by a simple modification of the well-known *Apriori* algorithm [1].

Since the existence of an item in a transaction is indicated by a probability, an advantage of the existential uncertain data model is that it allows more information to be captured by the dataset. Consider again the example patient dataset. If we adopt a binary data model, then each symptom/illness can either be present (1) or absent (0) in a patient record. Under the binary model, data analysts will be forced to set a threshold value for each symptom/illness to quantize the probabilities into either 1 or 0. In other words, information about those (marginally) low values is discarded. The uncertain data model, however, allows such information be retained and be available for analysis. The disadvantage of retaining such information is that the size of the dataset would be much larger

than that under the quantized binary model. This is particularly true if most of the existential probabilities are very small. Consequently, mining algorithms will run a lot slower on such large datasets. In this paper we propose an efficient technique for mining existential uncertain datasets, which exploit the statistical properties of low-valued items. Through experiments, we will show that the proposed technique is very efficient in terms of both CPU cost and I/O cost.

The rest of this paper is organized as follows. Section 2 describes the Possible Worlds interpretation of existential uncertain data and defines the *expected support* measure. Section 3 discusses a simple modification of the *Apriori* algorithm to mine uncertain data and explains why such a modification does not lead to an efficient algorithm. Section 4 presents a *data trimming* technique to improve mining efficiency. Section 5 presents some experimental results and discusses some observations. We conclude the study in Section 6.

2 Problem Definition

In our data model, an uncertain dataset D consists of d transactions $t_1, \ldots, t_d$. A transaction t_i contains a number of items. Each item x in t_i is associated with a *non-zero* probability $P_{t_i}(x)$, which indicates the likelihood that item x is present in transaction t_i. There are thus two possibilities of the world. In one case, item x is present in transaction t_i; in another case, item x is not in t_i. Let us call these two possibilities the two possible worlds, W_1 and W_2, respectively. We do not know which world is the real world but we do know, from the dataset, the probability of each world being the true world. In particular, if we let $P(W_i)$ be the probability that world W_i being the true world, then we have $P(W_1) = P_{t_i}(x)$ and $P(W_2) = 1 - P_{t_i}(x)$. We can extend this idea to cover cases in which transaction t_i contains other items. For example, let item y be another item in t_i with probability $P_{t_i}(y)$. If the observation of item x and item y are independently done[1], then there are four possible worlds. The probability of the world in which t_i contains both items x and y, for example, is $P_{t_i}(x) \cdot P_{t_i}(y)$. We can further extend the idea to cover datasets that contains more than one transaction. Figure 1 illustrates the 16 possible worlds derived from the patient records shown in Table 1. In traditional frequent itemset mining, the support count of an itemset X is defined as the number of transactions that contain X. For an uncertain dataset, such a support value is undefined since we do not know in the real world whether a transaction contains X with certainty. We can, however, determine the support of X with respect to any given possible world. Let us consider the worlds shown in Figure 1, the supports of itemset AB in world W_1 and W_6 are 2 and 1, respectively. If we can determine the probability of each possible world and the support of an itemset X in each world, we can determine the *expected support* of X.

Definition 1. *An itemset X is frequent if and only if its expected support not less than $\rho_s \cdot d$, where ρ_s is a user-specified support threshold.*

[1] For example, we can consider that different symptoms are diagnosed by independent medical tests.

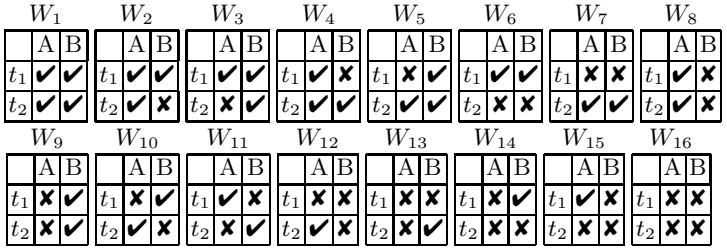

Fig. 1. 16 Possible Worlds derived from dataset with 2 transactions and 2 items

Given a world W_i and an itemset X, let us define $P(W_i)$ be the probability of world P_i and $S(X, W_i)$ be the support count of X in world W_i. Furthermore, we use $T_{i,j}$ to denote the set of items that the jth transaction, i.e., t_j, contains in the world W_i. If we assume that items' existential probabilities in transactions are determined through independent observations[2], then $P(W_i)$ and the expected support $S_e(X)$ of X are given by the following formulae:

$$P(W_i) = \prod_{j=1}^{d} \left(\prod_{x \in T_{i,j}} P_{t_j}(x) \cdot \prod_{y \notin T_{i,j}} (1 - P_{t_j}(y)) \right), \text{ and} \tag{1}$$

$$S_e(X) = \sum_{i=1}^{|W|} P(W_i) \times S(X, W_i). \tag{2}$$

where W is the set of possible worlds derived from an uncertain dataset D.

Computing $S_e(X)$ according to Equation 2 requires enumerating all possible worlds and finding the support count of X in each world. This is computationally infeasible since there are 2^m possible worlds where m is the total number of items that occur in all transactions of D. Fortunately, we can show that

$$S_e(X) = \sum_{j=1}^{|D|} \prod_{x \in X} P_{t_j}(x). \tag{3}$$

Thus, $S_e(X)$ can be computed by a single scan through the dataset D.

Proof. Let $S^{t_j}(X, W_i)$ be the support of X in transaction t_j w.r.t. possible world W_i. If $X \subseteq T_{i,j}$, $S^{t_j}(X, W_i) = 1$; otherwise, $S^{t_j}(X, W_i) = 0$.

$$\begin{aligned} S_e(X) &= \sum_{i=1}^{|W|} P(W_i) S(X, W_i) = \sum_{i=1}^{|W|} P(W_i) \sum_{j=1}^{|D|} S^{t_j}(X, W_i) \\ &= \sum_{j=1}^{|D|} \sum_{i=1}^{|W|} P(W_i) S^{t_j}(X, W_i) = \sum_{j=1}^{|D|} \sum_{X \subseteq T_{i,j}} P(W_i) = \sum_{j=1}^{|D|} \prod_{x \in X} P_{t_j}(x). \end{aligned}$$

[2] For example, the existential probabilities of two symptoms of the same patient are determined independently by two lab tests.

3 Preliminaries

Most of the algorithms devised to find frequent patterns (or itemsets) from conventional transaction datasets are based on the *Apriori* algorithm [1]. The algorithm relies on a property that all supersets of an infrequent itemset must not be frequent. Apriori operates in a bottom-up and iterative fashion. In the k^{th} iteration, the *Apriori-Gen* procedure generates all size-k candidate itemsets C^k and uses a *Subset-Function* procedure to verify their support counts. Candidate itemsets with support counts larger than a user-specified support threshold are regarded as frequent. The set of frequent k-itemsets L^k is then used by the Apriori-Gen procedure to generate candidates for next iteration. The algorithm terminates when C^{k+1} is empty.

Under our uncertainty model, the Subset-Function procedure has to be revised such that it can obtain the expected support count of each candidate. In the traditional Apriori algorithm, Subset-Function processes one transaction at a time by enumerating all size-k subsets contained in the transaction in the k^{th} iteration. The support count of a candidate is incremented by 1 if it is in C^k. By Equation 3, we will instead increment the expected support count by the product of the existential probabilities of all items $x \in X$. This modified algorithm is called the **U-Apriori** algorithm.

Inherited from the Apriori algorithm, U-Apriori does not scale well on large datasets. The poor efficiency problem becomes more serious under uncertain datasets, as mentioned in Section 1, in particular when most of the existential probabilities are of low values. Let us consider a transaction t containing three items A, B and C with existential probabilities $5\%, 0.5\%$ and 0.1%, respectively. In the Subset-Function procedure, the product of the probabilities ($0.05 \times 0.005 \times 0.001 = 0.00000025$) will be computed and the support count of candidate $\{ABC\}$ will be retrieved. By Equation 3, the support count of candidate $\{ABC\}$ should be incremented by 0.00000025 which is insignificantly small. If most of the existential probabilities are small, such insignificant increments will dominate the Subset-Function procedure and waste computational resources since in most cases an infrequent candidate will not be recognized as infrequent until most of the transactions are processed.

To illustrate the impact of items with low existential probabilities on the performance of U-Apriori, we conducted a preliminary experiment on five datasets. The datasets have the same set of frequent itemsets but are fine tuned to have different percentages of items with low existential probabilities. Let R be the percentage of items with low probabilities in a dataset. In the five datasets, R is set as $0\%, 33.3\%, 50\%, 66.6\%$ and 75% respectively. [3] In Figure 2a, we see that U-Apriori takes different amount of time to execute even though all datasets contain the same sets of frequent itemsets. We can conclude that when there are more items with low existential probabilities (larger R), U-Apriori becomes more inefficient. This result also indicates that by reducing the number of insignificant candidate increments, we might be able to reduce the execution time on all

[3] Please refer to Section 5 for the details of our data generation process.

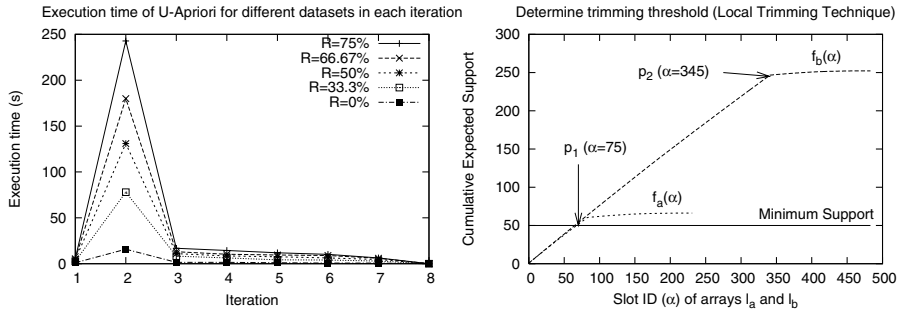

Fig. 2. a) The execution time of U-Apriori in different datasets. **b)** Cumulative expected support of the sorted arrays l_a and l_b of items a and b.

datasets to the time of the dataset with $R = 0\%$. This motivates our study on an efficient technique called *Data trimming* by exploiting the statistical properties of those items with low existential probabilities.

4 Data Trimming

To improve the efficiency of the U-Apriori algorithm, we propose a *data trimming* technique to avoid insignificant candidate support increments performed in the Subset-Function. The basic idea is to trim away items with low existential probabilities from the original dataset and to mine the trimmed dataset instead. Hence the computational cost of those insignificant candidate increments can be reduced. In addition, the I/O cost can be greatly reduced since the size of the trimmed dataset is much smaller than the original one.

Data Trimming Framework. More specifically, the data trimming technique works under a framework that consists three modules: the *trimming module*,

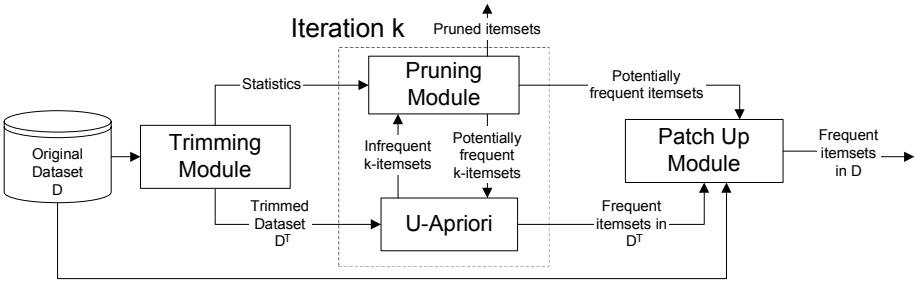

Fig. 3. The Data Trimming Framework

pruning module and *patch up module*. As shown in figure 3, the mining process starts by passing an uncertain dataset D into the *trimming module*. It first obtains the frequent items by scanning D once. A trimmed dataset D^T is constructed by removing the items with existential probabilities smaller than a trimming threshold ρ_t in the second iteration. Depending on the trimming strategy, ρ_t can be either global to all items or local to each item. Some statistics such as the maximum existential probability being trimmed for each item is kept for error estimation.

D^T is then mined by U-Apriori. Notice that if an itemset is frequent in D^T, it must also be frequent in D. On the other hand, if an itemset is infrequent in D^T, we cannot conclude that it is infrequent in D.

Definition 2. *An itemset X is potentially frequent if $S_e^T(X) \leq d\rho_s \leq S_e^T(X) + e(X)$ where $S_e^T(X)$ is the expected support of X in D^T and $e(X)$ is the upper bound of the error estimated for $S_e^T(X)$.*

Lemma 1. *An itemset X cannot be frequent if $S_e^T(X) + e(X) < d\rho_s$.*

The role of the *pruning module* is to estimate the upper bound of the mining error $e(X)$ by the statistics gathered from the *trimming module* and to prune the itemsets which cannot be frequent in D according to Lemma 1. After mining D^T, the expected supports of the frequent and potentially frequent itemsets are verified against the original dataset D by the *patch up module*.

A number of trimming, pruning and patch up strategies can be used under this framework. Due to limitation of space, we only present a simple method, called the *Local trimming, Global pruning and Single-pass patch up strategy* (the **LGS-Trimming** strategy), in this paper.

Local Trimming Strategy. The *Local trimming* strategy uses one trimming threshold $\rho_t(x)$ for each item x. $\rho_t(x)$ can be determined based on the distribution of existential probabilities of x in D. The distribution can be obtained by sorting the existential probabilities of x in D in descending order and putting them in an array l_x. We then plot the curve $f_x(\alpha) = \sum_{i=0}^{\alpha} l_x[i]$ where the y-axis is the cumulative sum of the probabilities $\sum_{i=0}^{\alpha} l_x[i]$ and the x-axis is the slot ID of l_x. Figure 2b shows the curves $f_a(\alpha)$ and $f_b(\alpha)$ of two hypotheoretical items a and b. The horizontal line labeled "minsup" is the minimum support threshold.

We regard item a as *marginally frequent* because $S_e(a)$ exceeds the minimum support by a small fraction (e.g. $d\rho_s \leq S_e(a) \leq 1.1 \times d\rho_s$). Assume $f_a(\alpha)$ intersects with the minimum support line at about $\alpha = i$. In this case, the Local trimming strategy sets the trimming threshold $\rho_t(a)$ to be $l_x[i]$, which is the existential probability of the item at the ith slot of the array l_x (i.e. $\rho_t(a) = l_a[75]$). The rationale is that the supersets of a are likely to be infrequent, therefore those insignificant candidate increments with existential probabilities smaller than $\rho_t(a)$ are likely to be redundant.

On the other hand, we classify item b as another type of items as $S_e(b) \gg d\rho_s$. The Local trimming strategy determines $\rho_t(b)$ based on the change of slope of $f_b(\alpha)$. In this case, since the chance of the supersets of b to be frequent is larger,

we adopt a more conservative approach. We use point p_2 in Figure 2b as a reference point to determine $\rho_t(b)$ (i.e. $\rho_t(b) = l_b[345]$)[4]. The reason is that if one of the supersets of b is actually infrequent, the error would be small enough for the Pruning module to obtain a tight estimation and identify it as an infrequent itemset by Lemma 1.

Global Pruning Strategy. We illustrate the *Global pruning* strategy by an example in the second iteration. Let $M^T(x)$ be the maximum of the existential probabilities of those untrimmed item x, and similarly $M^{\sim T}(x)$ for those trimmed x. We also let $S_e^T(x)$ be the sum of the existential probabilities of those untrimmed x, and similarly $S_e^{\sim T}(x)$ for those trimmed x. If an itemset {AB} is infrequent in D^T (i.e. $S_e^T(AB) < d\rho_s$), we can obtain the upper bound of the error $e(AB)$ by the following formula:

$$e(AB) = \hat{S}_e^{T,\sim T}(AB) + \hat{S}_e^{\sim T,T}(AB) + \hat{S}_e^{\sim T,\sim T}(AB). \tag{4}$$

where $\hat{S}_e^{T,\sim T}(AB)$ is an upper bound estimation of the expected support of {AB} for all transactions t with $P_t(A) \geq \rho_t(A)$ and $P_t(B) < \rho_t(B)$.

If we assume all the untrimmed items A exist with maximum existential probability $M^T(A)$, then the maximum number of transactions with an untrimmed item A which may coexist with a trimmed item B is given by $\frac{S_e^T(A) - S_e^T(AB)}{M^T(A)}$. On the other hand, if we assume all the trimmed items B exist with maximum probability $M^{\sim T}(B)$, then the maximum number of transactions with a trimmed item B is given by $\frac{S_e^{\sim T}(B)}{M^{\sim T}(B)}$. Therefore, we can obtain $\hat{S}_e^{T,\sim T}(AB)$ as shown in Equation 5. $\hat{S}_e^{\sim T,T}(AB)$ is similarly obtained.

$\hat{S}_e^{\sim T,\sim T}(AB)$ is an upper bound estimation of the expected support of {AB} for all transactions t with $P_t(A) < \rho_t(A)$ and $P_t(B) < \rho_t(B)$, assuming that the case of estimating $\hat{S}_e^{T,\sim T}(AB)$ and $\hat{S}_e^{\sim T,T}(AB)$ happens in D. It can be calculated by Equation 7 after obtaining $\hat{S}_e^{T,\sim T}(AB)$ and $\hat{S}_e^{\sim T,T}(AB)$.

$$\hat{S}_e^{T,\sim T}(AB) = min(\frac{S_e^T(A) - S_e^T(AB)}{M^T(A)}, \frac{S_e^{\sim T}(B)}{M^{\sim T}(B)}) \cdot M^T(A) \cdot M^{\sim T}(B). \tag{5}$$

$$\hat{S}_e^{\sim T,T}(AB) = min(\frac{S_e^{\sim T}(A)}{M^{\sim T}(A)}, \frac{S_e^T(B) - S_e^T(AB)}{M^T(B)}) \cdot M^{\sim T}(A) \cdot M^T(B). \tag{6}$$

$$\hat{S}_e^{\sim T,\sim T}(AB) = min(\frac{S_e^{\sim T}(A) - \hat{S}_e^{\sim T,T}(AB)}{M^{\sim T}(A)}, \frac{S_e^{\sim T}(B) - \hat{S}_e^{T,\sim T}(AB)}{M^{\sim T}(B)}) \cdot M^{\sim T}(A) \cdot M^{\sim T}(B). \tag{7}$$

Single-pass Patch Up Strategy. The *Single-pass patch up* strategy requires only one scan on the original dataset D. This strategy requires the *Apriori-Gen*

[4] Due to space limitation, we only present the abstract idea of Local trimming strategy in this paper.

procedure to include the potentially frequent itemsets during the mining process so that the set of potentially frequent itemsets will not miss any real frequent itemsets. In the patch up phase, the true expected supports of potentially frequent itemsets are verified by a single scan on the original dataset D. At the same time, the true expected supports of frequent itemsets in D^T are also recovered.

5 Experimental Evaluation

We ran our experiments on Linux Kernel version 2.6.10 machine with 1024 MB of memory. The U-Apriori algorithm and the LGS-Trimming technique were implemented using C programming language.

Data were generated in the following two-step procedure. First we generate data without uncertainty using the IBM synthetic generator used in [1]. This step is to generate dataset which contains frequent itemsets. We set the average number of items per transaction (T_{high}) to be 20, the average length of frequent itemsets (I) to be 6 and the number of transactions (D) to be 100K [5].

In the second step, we introduce uncertainty to each item of the dataset generated from the first step. Since we want to maintain the frequent patterns hidden in the dataset, we assign each items with relatively high probabilities following a normal distribution with specified mean HB and standard deviation HD. To simulate items with low probabilities, we add T_{low} number of items into each transaction. These items have probabilities in normal distribution with mean LB and standard deviation LD. Therefore, the average number of items per transaction, denoted as T, is equal to $T_{high} + T_{low}$. We use R to denote the percentage of items with low probabilities in the dataset (i.e. $R = \frac{T_{low}}{T_{high}+T_{low}}$).

As an example, $T80R75I6D100KHB90HD5LB10LD6$ represents an uncertain dataset with 80 items per transaction on average. Of the 80 items, 20 items are assigned with high probabilities and 60 items are assigned with low probabilities. The high(low) probabilities are generated following normal distribution with mean equal to 90%(10%) and standard deviation equal to 5%(6%). For simplicity, we call this dataset *Synthetic-1* in later sections.

5.1 Varying Number of Low Probability Items Per Transaction

We first investigate the CPU cost of U-Apriori and LGS-Trimming on datasets with different number of low probability items per transaction. We keep the same set of frequent itemsets in all datasets, therefore an increase in R means more low-probability items are added during the second step of data generation. We set $\rho_s = 0.5\%$ in the experiments. Figure 4a and 4b show the CPU cost and the percentage of CPU cost saving (compare with U-Apriori) of U-Apriori and LGS-Trimming as R varies from 0% to 90%.

[5] We have conducted our experiments using different values of T_{high}, I and D but due to the space limitation we only report a representative result using $T_{high}20I6D100K$ in this paper.

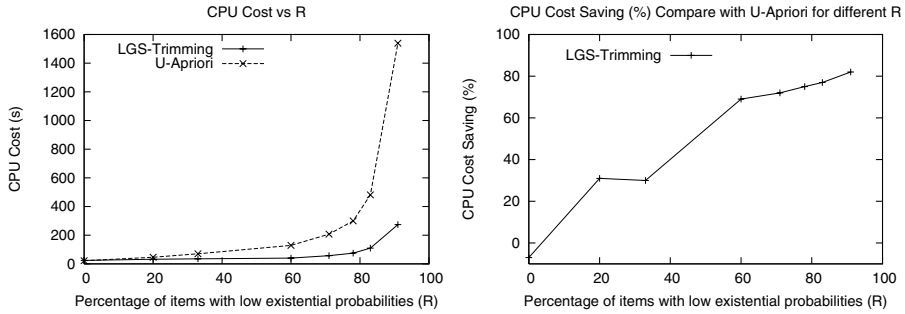

Fig. 4. CPU cost and saving with different R

From Figures 4a, we observe that the CPU cost of U-Apriori increases exponentially with the percentage of low probability items in the dataset. This is mainly due to the combinatorial explosion of subsets contained in a transaction. This leads to huge amounts of insignificant candidate support increments in the Subset-Function. For instance, when R is 90%, the average number of items per transaction with non-zero probability is about 200 (20 items with high probabilities, 180 items with low probabilities), leading to ${}_{200}C_2 = 19900$ size-2 subsets per transaction. In other words, there are 19900 candidate searches per transaction in the second iteration. When R is 50%, however, there are only ${}_{40}C_2 = 780$ candidate searches per transaction in the second iteration.

From Figure 4b, we see that the LGS-Trimming technique achieves positive CPU cost saving when R is over 3%. It achieves more than 60% saving when R is 50% or larger. When R is too low, fewer low probability items can be trimmed and the saving cannot compensate with the extra computational effort in the patch up phase. These figures suggest that the LGS-Trimming technique is very scalable to the percentage of low probability items in the dataset.

5.2 Varying Minimum Support Threshold

This section assesses the performance of U-Apriori and LGS-Trimming by varying ρ_s from 1% to 0.1%. Here we only report the result of using *Synthetic-1* in this experiment because experimental results on other datasets with different values of HB, HD, LB and LD also lead to a similar conclusion. Figures 5a and 5b show the CPU cost and the saving (in %) of the two mining techniques.

Figure 5a shows that LGS-Trimming outperforms U-Apriori for all values of ρ_s. Figure 5b shows that LGS-Trimming achieves very high and steady CPU cost saving ranging from 60% to 80%. The percentage of CPU cost saving increases gently when ρ_s increases because the low probability items become less significant to the support of itemsets when the support threshold increases. Therefore more low probability items can be trimmed, leading to better saving.

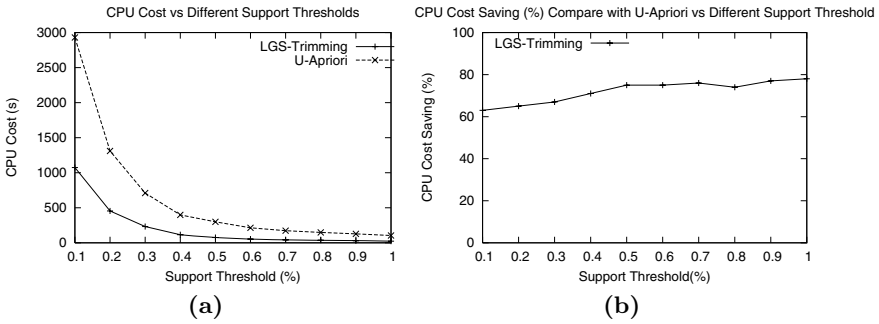

Fig. 5. CPU cost and saving with different ρ_s

5.3 CPU Cost and I/O Cost in Each Iteration

In this section we compare the CPU and I/O cost in each iteration of U-Apriori and LGS-Trimming. The dataset we use is *Synthetic-1* and we set $\rho_s = 0.5\%$. From Figure 6a, we see that the CPU cost of LGS-Trimming is smaller than U-Apriori from the second to the second last iteration. In particular, LGS-Trimming successfully relieves the computational bottleneck of U-Apriori and achieves over 90% saving in the second iteration. In the first iteration, the CPU cost of LGS-Trimming is slightly larger than U-Apriori because extra effort is spent on gathering statistics for the trimming module to trim the original dataset. Notice that iteration 8 is the patch up iteration which is the overhead of the LGS-Trimming algorithm. These figures show that the computational overhead of LGS-Trimming is compensated by the saving from the second iteration.

Figure 6b shows the I/O cost in terms of dataset scan (with respect to the size of the original dataset) in each iteration. We can see that I/O saving starts from iteration 3 to the second last iteration. The extra I/O cost in the second

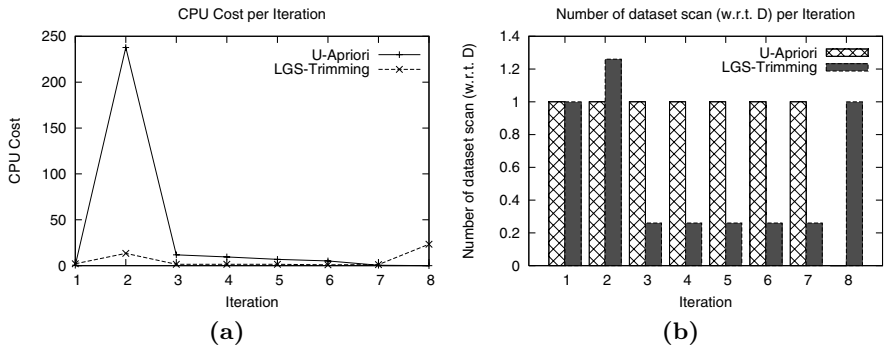

Fig. 6. CPU and I/O costs of U-Apriori and LGS-Trimming in each iteration

iteration is the cost of creating the trimmed dataset. In this case, LGS-Trimming reduces the size of the original dataset by a factor of 4 and achieves 35% I/O cost saving in total. As U-Apriori iterates k times to discover a size-k frequent itemset, longer frequent itemsets favors LGS-Trimming and the I/O cost saving will be more significant.

6 Conclusions

In this paper we studied the problem of mining frequent itemsets from existential uncertain data. We introduced the U-Apriori algorithm, which is a modified version of the Apriori algorithm, to work on such datasets. We identified the computational problem of U-Apriori and proposed a data trimming framework to address this issue. We proposed the LGS-Trimming technique under the framework and verified, by extensive experiments, that it achieves very high performance gain in terms of both computational cost and I/O cost. Unlike U-Apriori, LGS-Trimming works well on datasets with high percentage of low probability items. In some of the experiments, LGS-Trimming achieves over 90% CPU cost saving in the second iteration of the mining process, which is the computational bottleneck of the U-Apriori algorithm.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of 20^{th} International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, Morgan Kaufmann (1994) 487–499
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proc. of the 11^{th} International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, IEEE Computer Society (1995) 3–14
3. Dai, X., Yiu, M.L., Mamoulis, N., Tao, Y., Vaitis, M.: Probabilistic spatial queries on existentially uncertain data. In: SSTD. Volume 3633 of Lecture Notes in Computer Science., Springer (2005) 400–417
4. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: KDD. (1998) 80–86
5. Rushing, A., Ranganath, S., Hinke, H., Graves, J.: Using association rules as texture features. IEEE Trans. Pattern Anal. Mach. Intell. **23**(8) (2001) 845–858
6. Zimányi, E., Pirotte, A.: Imperfect information in relational databases. In: Uncertainty Management in Information Systems. (1996) 35–88