A Decremental Approach for Mining Frequent Itemsets from Uncertain Data[⋆]

Chun-Kit Chui and Ben Kao

Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
{ckchui,kao}@cs.hku.hk

Abstract. We study the problem of mining frequent itemsets from *uncertain data* under a *probabilistic model*. We consider transactions whose items are associated with *existential probabilities*. A *decremental pruning* (DP) technique, which exploits the statistical properties of items' existential probabilities, is proposed. Experimental results show that DP can achieve significant computational cost savings compared with existing approaches, such as U-Apriori and LGS-Trimming. Also, unlike LGS-Trimming, DP does not require a user-specified trimming threshold and its performance is relatively insensitive to the population of low-probability items in the dataset.

1 Introduction

Frequent itemset mining (FIM) is a core component in many data analysis tasks such as association analysis [1] and sequential-pattern mining [2]. Traditionally, FIM is applied to data that is certain and precise. As an example, a transaction in a market-basket dataset registers items that are purchased by a customer. Applying FIM on such a dataset allows one to identify items that are often purchased together. In this example, the presence/absence of an item in a transaction is known with certainty. Existing FIM algorithms, such as the well-known Apriori algorithm [1] and other variants, were designed for mining "certain" data.

Most of the previous studies on FIM assume a data model under which transactions capture doubtless facts about the items that are contained in each transaction. However, in many applications, the existence of an item in a transaction is best captured by a probability. As an example, consider experiments that test certain drug-resistant properties of pathogens. Results of such tests can be represented by a transactional dataset: each pathogen is represented by a transaction and the drugs it shows resistance to are listed as items in the transaction. Applying FIM on such a dataset allows us to discover multi-drug-resistant associations [3]. In practice, due to measurement and experimental errors, multiple measurements or experiments are conducted to obtain a higher confidence of the

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

results. In such cases, the existence of an item or property in a transaction should be expressed in terms of a probability. For example, if Streptococcus Pneumoniae (a pathogen) shows resistance to penicillin (an antibiotics drug) 90 times out of 100 experiments, the probability that the property "penicillin-resistant" *exists* in Streptococcus Pneumoniae is 90%. We call this kind of probability *existential probability.* In this paper we study the problem of applying FIM on datasets under the *existential uncertain data model*, in which each item is associated with an existential probability that indicates the likelihood of its presence in a transaction. Table 1 shows an example of an existential uncertain dataset.

Table 1. An existential uncertain dataset with 2 transactions t_1, t_2 and 2 items a, b

Transaction \ Item	a	b
t_1	90%	80%
t_2	40%	70%

The problem of mining frequent itemsets under the existential uncertain data model was first studied in [4]. The Apriori algorithm was modified to mine uncertain data. The modified algorithm, called U-Apriori, was shown to be computationally inefficient. A data trimming framework (LGS-Trimming) was proposed to reduce the computational and I/O costs of U-Apriori. As a summary, given an existential uncertain dataset D, LGS-Trimming creates a trimmed dataset D^T by removing items with low existential probabilities in D. The trimming framework works under the assumption that a non-trivial portion of the items in the dataset are associated with low existential probabilities (e.g., a pathogen may be highly resistant to a few drugs but not so for most of the others). Based on this assumption, the size of D^T is significantly smaller than D and mining D^T instead of D has the following advantages:

- The I/O cost of scanning D^T is smaller.
- Since many low-probability items have been removed, transactions in D^T are much smaller. Hence, there are a lot fewer subsets contained in transactions leading to much faster subset testing of candidate itemsets and faster support counting.

However, there are disadvantages of the trimming framework. First, there is the overhead of creating D^T. Second, since D^T is incomplete information, the set of frequent itemsets mined from it is only a subset of the complete set. A patch-up phase (and thus some overhead) is therefore needed to recover those missed frequent itemsets. As a result, if there are relatively few low-probability items in D, then D^T and D will be of similar sizes. The savings obtained by LGS-Trimming may not compensate for the overhead incurred. The performance of LGS-Trimming is thus sensitive to the percentage (R) of items with low existential probabilities. Trimming can be counter-productive when R is very low. Third, a trimming threshold ρ_t (to determine "low" probability) is needed, which in some cases could be hard to set. A large ρ_t implies a greater reduction of the size of D but a larger overhead in the patch-up phase to recover missed frequent itemsets. On the other hand, a small ρ_t would trim D by little extent resulting in little savings. The performance of Trimming is thus sensitive to ρ_t. In [4], it was assumed that the existential probabilities of items in a dataset follow a *bimodal distribution*. That is, most items' can be classified as very-high-probability ones or very-low-probability ones. There were few items with moderate existential probabilities. In that case, it is easy to determine ρ_t as there is a clearcut distinction between high and low existential probabilities. It would be harder to select an appropriate ρ_t if the distribution of existential probabilities is more uniform.

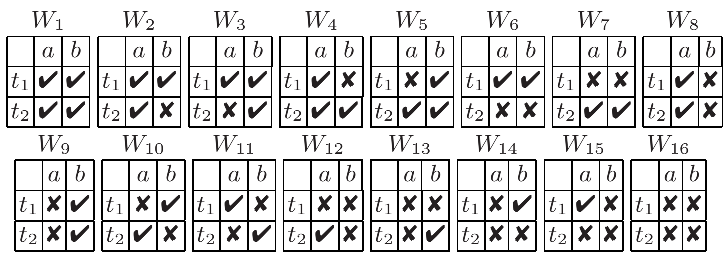
In this paper we propose an alternative method, called *Decremental Pruning* (DP), for mining frequent itemsets from existential uncertain data. As we will discuss in later sections, DP exploits the statistical properties of existential probabilities to gradually reduce the set of candidate itemsets. This leads to more efficient support counting and thus significant CPU cost savings. Comparing with LGS-Trimming, DP has two desirable properties: (1) it does not require a user-specified trimming threshold; (2) its performance is relatively less sensitive to R, the fraction of small-probability items in the dataset. DP is thus more applicable to a larger range of applications. Moreover, we will show that DP and LGS-Trimming are complementary to each other. They can be combined to achieve an even better performance.

The rest of this paper is organized as follows. Section 2 describes the mining problem and revisits the brute force U-Apriori algorithm. Section 3 presents the DP approach. Section 4 presents some experimental results and discusses some observations. We conclude the study in Section 5.

2 Preliminaries

In the existential uncertain data model, a 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[1]. A *Possible World* model [5] can be applied to interpret an existential uncertain dataset. Basically, each probability $P_{t_i}(x)$ associated with an item x derives two possible worlds, say, W_1 and W_2. In World W_1, item x is present in transaction t_i; In World W_2, item x is not in t_i. Let $P(W_j)$ be the probability that World W_j being the true world, then we have $P(W_1) = P_{t_i}(x)$ and $P(W_2) = 1 - P_{t_i}(x)$. This idea can be extended to cover cases in which transaction t_i contains other items. For example, let y be another item in t_i with probability $P_{t_i}(y)$. Assume that the observations of item x and item y are independently done, then there are four possible worlds. In particular, the probability of the world in which t_i contains both items x and y is $P_{t_i}(x) \cdot P_{t_i}(y)$. We can further generalize the idea to datasets that contain more than one transaction. Figure 1 illustrates the 16 possible worlds derived from the dataset shown in Table 1.

[1] If an item has 0 existential probability, it does not appear in the transaction.



W_1

	a	b
t_1	✓	✓
t_2	✓	✓

W_2

	a	b
t_1	✓	✓
t_2	✓	✗

W_3

	a	b
t_1	✓	✓
t_2	✗	✓

W_4

	a	b
t_1	✓	✗
t_2	✓	✓

W_5

	a	b
t_1	✗	✓
t_2	✓	✓

W_6

	a	b
t_1	✓	✓
t_2	✗	✗

W_7

	a	b
t_1	✗	✗
t_2	✓	✓

W_8

	a	b
t_1	✓	✗
t_2	✓	✗

W_9

	a	b
t_1	✗	✓
t_2	✗	✓

W_{10}

	a	b
t_1	✗	✓
t_2	✓	✗

W_{11}

	a	b
t_1	✓	✗
t_2	✗	✓

W_{12}

	a	b
t_1	✗	✗
t_2	✓	✗

W_{13}

	a	b
t_1	✗	✗
t_2	✗	✓

W_{14}

	a	b
t_1	✗	✓
t_2	✗	✗

W_{15}

	a	b
t_1	✓	✗
t_2	✗	✗

W_{16}

	a	b
t_1	✗	✗
t_2	✗	✗

Fig. 1. 16 possible worlds derived from dataset with 2 transactions and 2 items

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 set containment is probabilistic. However, we note that each possible world derived from an uncertain dataset is certain, and therefore support counts are well-defined with respect to each world. For example, the support counts of itemset $\{a, b\}$ in Worlds W_1 and W_6 (Figure 1) are 2 and 1, respectively. In [4], the notion of *expected support* was proposed as a frequency measure. Let W be the set of all possible worlds derivable from an uncertain dataset D. Given a world $W_j \in W$, let $P(W_j)$ be the probability of World W_j; $S(X, W_j)$ be the support count of X with respect to W_j; and $T_{i,j}$ be the i^{th} transaction in World W_j. Assuming that items' existential probabilities are determined through independent observations, then $P(W_j)$ and the expected support $S_e(X)$ of an itemset X are given by the following formulae[2]:

$$P(W_j) = \prod_{i=1}^{|D|} \left(\prod_{x \in T_{i,j}} P_{t_i}(x) \cdot \prod_{y \notin T_{i,j}} (1 - P_{t_i}(y)) \right), \quad \text{and} \tag{1}$$

$$S_e(X) = \sum_{j=1}^{|W|} P(W_j) \times S(X, W_j) = \sum_{i=1}^{|D|} \prod_{x \in X} P_{t_i}(x). \tag{2}$$

Problem Statement. Given an existential uncertain dataset D and a user-specified support threshold ρ_s, the problem of mining frequent itemsets is to return all itemsets X with expected support $S_e(X) \geq \rho_s \cdot |D|$.

U-Apriori, a modified version of the Apriori algorithm, was presented in [4] as a baseline algorithm to solve the problem. The difference between Apriori and U-Apriori lies in the way supports are counted. Given a candidate itemset X and a transaction t_i, Apriori tests whether $X \subseteq t_i$. If so, the support count of X is incremented by 1. Under U-Apriori, the support count of X is incremented by the value $\prod_{x \in X} P_{t_i}(x)$ instead (see Equation 2).

[2] Readers are referred to [4] for the details of the derivations.

3 Decremental Pruning

In this section we describe the *Decremental Pruning* (DP) technique, which exploits the statistical properties of the existential probabilities of items to achieve candidate reduction during the mining process. The basic idea is to estimate upper bounds of candidate itemsets' expected supports progressively after each dataset transaction is processed. If a candidate's upper bound falls below the support threshold ρ_s, the candidate is immediately pruned. To illustrate, let us consider a sample dataset shown in Table 2. Assume a support threshold $\rho_s = 0.5$, the minimum support count is $min\_sup = 4 \times 0.5 = 2$. Consider the candidate itemset $\{a, b\}$. To obtain the expected support of $\{a, b\}$, denoted as $S_e(\{a, b\})$, U-Apriori scans the entire dataset once and obtains $S_e(\{a, b\}) = 1.54$, which is infrequent.

Table 2. An example of existentially uncertain dataset

Transaction \ Item	a	b	c	d
t_1	1	0.5	0.3	0.2
t_2	0.9	0.8	0.7	0.4
t_3	0.3	0	0.9	0.7
t_4	0.4	0.8	0.3	0.7

During the dataset scanning process, we observe that a candidate itemset X can be pruned before the entire dataset is scanned. The idea is to maintain a *decremental counter* $\hat{S}_e(X, X')$ for some non-empty $X' \subset X$. The counter maintains an upper bound of the expected support count of X, i.e., $S_e(X)$. This upper bound's value is progressively updated as dataset transactions are processed. We use $\hat{S}_e(X, X', k)$ to denote the value of $\hat{S}_e(X, X')$ after transactions $t_1, \ldots, t_k$ have been processed.

Definition 1. ***Decremental Counter.*** *For any non-empty* $X' \subset X$, $k \geq 0$, $\hat{S}_e(X, X', k) = \sum_{i=1}^{k} \prod_{x \in X} P_{t_i}(x) + \sum_{i=k+1}^{|D|} \prod_{x \in X'} P_{t_i}(x)$.

From Equation 2, we have

$$\begin{aligned}
S_e(X) &= \sum_{i=1}^{|D|} \prod_{x \in X} P_{t_i}(x) \\
&= \sum_{i=1}^{k} \prod_{x \in X} P_{t_i}(x) + \sum_{i=k+1}^{|D|} \prod_{x \in X} P_{t_i}(x) \\
&\leq \sum_{i=1}^{k} \prod_{x \in X} P_{t_i}(x) + \sum_{i=k+1}^{|D|} \left(\prod_{x \in X'} P_{t_i}(x) \cdot \prod_{x \in X - X'} 1 \right) \\
&= \hat{S}_e(X, X', k).
\end{aligned}$$

Hence, $\hat{S}_e(X, X', k)$ is an upper bound of $S_e(X)$. Essentially, we are assuming that the probabilities of all items $x \in X - X'$ are 1 in transactions $t_{k+1}, \ldots, t_{|D|}$ in estimating the upper bound. Also, $\hat{S}_e(X, X', 0) = \sum_{i=1}^{|D|} \prod_{x \in X'} P_{t_i}(x) = S_e(X')$.

In our running example, suppose we have executed the first iteration of U-Apriori and have determined the expected supports of all 1-itemsets, in particular, we know $S_e(\{a\}) = 2.6$. At the beginning of the 2^{nd} iteration, we have, for the candidate itemset $\{a, b\}$, $\hat{S}_e(\{a, b\}, \{a\}, 0) = S_e(\{a\}) = 2.6$. We then process the first transaction t_1 and find that $P_{t_1}(b)$ is 0.5 (instead of 1 as assumed when we calculated the upper bound), we know that we have overestimated $S_e(\{a, b\})$ by $P_{t_1}(a) \times (1 - P_{t_1}(b)) = 0.5$. Therefore, we refine the bound and get $\hat{S}_e(\{a, b\}, \{a\}, 1) = \hat{S}_e(\{a, b\}, \{a\}, 0) - 0.5 = 2.1$. Next, we process t_2. By similar argument, we know that we have overestimated the support by $0.9 \times (1 - 0.8) = 0.18$. We thus update the bound to get $\hat{S}_e(\{a, b\}, \{a\}, 2) = \hat{S}_e(\{a, b\}, \{a\}, 1) - 0.18 = 1.92$. At this point, the bound has dropped below the support threshold. The candidate $\{a, b\}$ is thus infrequent and can be pruned.

Equation 3 summarizes the initialization and update of the decremental counter $\hat{S}_e(X, X', k)$:

$$\hat{S}_e(X, X', k) = \begin{cases} S_e(X') & \text{if } k = 0; \\ \hat{S}_e(X, X', k-1) - S_e^{t_k}(X') \times \{1 - S_e^{t_k}(X - X')\} & \text{if } k > 0. \end{cases} \tag{3}$$

where $S_e^{t_k}(X') = \prod_{x \in X'} P_{t_k}(x)$ and $S_e^{t_k}(X - X') = \prod_{x \in X - X'} P_{t_k}(x)$.

From the example, we see that $\{a, b\}$ can be pruned before the entire dataset is scanned. This candidate reduction potentially saves a lot of computational cost. However, there are $2^{|X|} - 2$ non-empty proper subsets of a candidate itemset X. The number of decremental counters is thus huge. Maintaining a large number of decremental counters involves too much overhead, and the DP method could be counter-productive. We propose two methods for reducing the number of decremental counters while maintaining a good pruning effectiveness in the rest of this section.

Aggregate by Singletons (AS). The AS method reduces the number of decremental counters to the number of frequent singletons. First, only those decremental counters $\hat{S}_e(X, X')$ where X' is a frequent singleton are maintained. Second, given a frequent item x, the decremental counters $\hat{S}_e(X, \{x\})$ for any itemset X that contains x are replaced by a *singleton decremental counter* $d_s(x)$. Let $d_s(x, k)$ be the value of $d_s(x)$ after the first k data transactions have been processed. Equation 4 shows the initialization and update of $d_s(x, k)$.

$$d_s(x, k) = \begin{cases} S_e(\{x\}) & \text{if } k = 0; \\ d_s(x, k-1) - P_{t_k}(x) \times \{1 - max_s(k)\} & \text{if } k > 0. \end{cases} \tag{4}$$

where $max_s(k) = max\{P_{t_k}(x') | x' \in t_k, x' \neq x\}$ returns the maximum existential probability among the items (except x) in transaction t_k.

One can prove by induction that $\hat{S}_e(X, \{x\}, k) \leq d_s(x, k)$ for any itemset X that contains item x. With the AS method, the aggregated counters can be organized in an *array*. During the mining process, if a counter's value $d_s(x, k)$ drops below the support requirement, we know that any candidate itemset X that contains x must not be frequent and hence can be pruned. Also, we can remove item x from the dataset starting from transaction t_{k+1}. Therefore, AS not only achieves candidate reduction, it also shrinks dataset transactions. The latter allows more efficient subset testing during support counting.

Common-Prefix Method (CP). The CP method aggregates the decremental counters of candidates with common prefix. Here, we assume that items follow a certain ordering Φ, and the set of items of an itemset is listed according to Φ. First, only decremental counters of the form $\hat{S}_e(X, X')$ where X' is a proper prefix of X (denoted by $X' \sqsubset X$) are maintained. Second, given an itemset X', all counters $\hat{S}_e(X, X')$ such that $X' \sqsubset X$ are replaced by a *prefix decremental counter* $d_p(X')$. Let $d_p(X', k)$ be the value of $d_p(X')$ after the first k data transactions have been processed. Equation 5 shows the initialization and update of $d_p(X', k)$.

$$d_p(X', k) = \begin{cases} S_e(X') & \text{if } k = 0; \\ d_p(X', k-1) - S_e^{t_k}(X') \times \{1 - max_p(k)\} & \text{if } k > 0. \end{cases} \tag{5}$$

where $S_e^{t_k}(X') = \prod_{x \in X'} P_{t_k}(x)$ and $max_p(k) = max\{P_{t_k}(z) | z$ is after all the items in X' according to the item ordering $\Phi\}$.

Again, by induction, we can prove that $\hat{S}_e(X, X', k) \leq d_p(X', k)$ for any $X' \sqsubset X$. Hence when $d_p(X', k)$ drops below the support requirement, we can conclude that any candidate itemset X such that $X' \sqsubset X$ must be infrequent and can thus be pruned. We remark that since most of the traditional frequent itemset mining algorithms apply a prefix-tree data structure to organize candidates [1][6][4], the way that CP aggregates the decremental counters facilitates its integration with the prefix-tree data structure.

Figure 2 shows the size-2 candidates of the dataset in Table 2 organized in a hash-tree data structure [1]. A hash-tree is essentially a prefix tree, where candidates with the same prefix are organized under the same sub-tree. A prefix is thus associated with a node in the tree. A prefix decremental counter $d_p(X')$ is stored in the parent node of the node that is associated with the prefix X'. For example, $d_p(b)$ is stored in the root node since the prefix b is at level 1 of the tree (the second child node shown in Figure 2). [1] presented a recursive strategy for searching candidates that are contained in each transaction using a hash-tree structure. We illustrate the steps of processing a transaction t_1 from our running example (see Table 2) and explain how the counter $d_p(b)$ is updated in Figure 2.

From the figure, we see that $d_p(b, 1) = 1.75$ after t_1 is processed. Since $d_p(b, 1)$ is an upper bound of the expected supports of $\{b, c\}$ and $\{b, d\}$, and since $d_p(b, 1)$ is smaller than the support requirement, we conclude that both $\{b, c\}$ and $\{b, d\}$ are infrequent and are thus pruned. With the hash-tree structure, we can virtually prune the candidates by setting the pointer $root.hash(b) = NULL$. Also, the counter $d_p(b)$ is removed from the root. As a result, the two candidates cannot

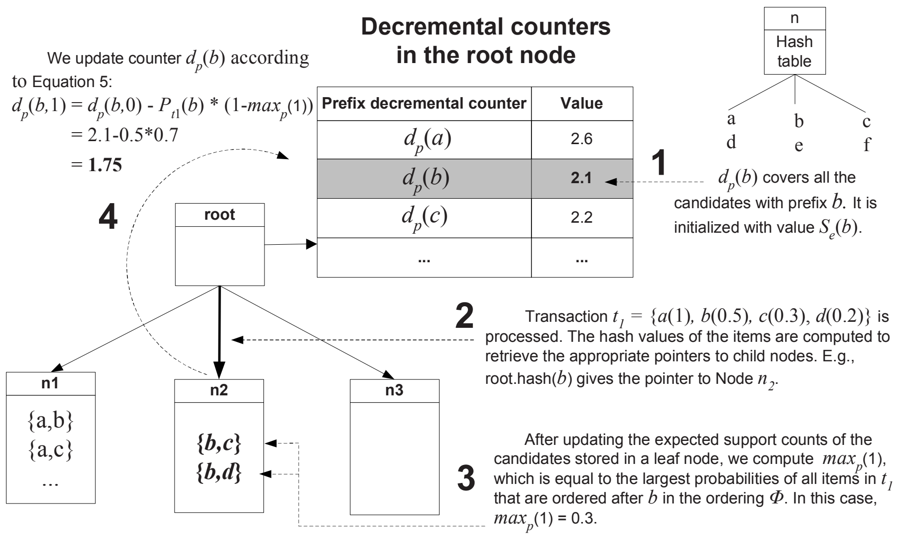

Decremental counters in the root node
We update counter $d_p(b)$ according to Equation 5:
$d_p(b,1) = d_p(b,0) - P_{t1}(b) * (1-max_p(1))$
$= 2.1-0.5*0.7$
$= \mathbf{1.75}$
Prefix decremental counter
Value
$d_p(a)$ 2.6
$d_p(b)$ 2.1
$d_p(c)$ 2.2
... ...
n
Hash table
a d
b e
c f
1
$d_p(b)$ covers all the candidates with prefix b. It is initialized with value $S_e(b)$.
4
root
2
Transaction $t_1 = \{a(1), b(0.5), c(0.3), d(0.2)\}$ is processed. The hash values of the items are computed to retrieve the appropriate pointers to child nodes. E.g., root.hash(b) gives the pointer to Node n_2.
n1
{a,b}
{a,c}
...
n2
{b,c}
{b,d}
n3
3
After updating the expected support counts of the candidates stored in a leaf node, we compute $max_p(1)$, which is equal to the largest probabilities of all items in t_1 that are ordered after b in the ordering Φ. In this case, $max_p(1) = 0.3$.

Fig. 2. A size-2 candidate hash tree with prefix decremental counters

be reached when subsequent transactions are processed. The computational cost of incrementing the expected support counts of the two candidates in subsequent transactions is saved.

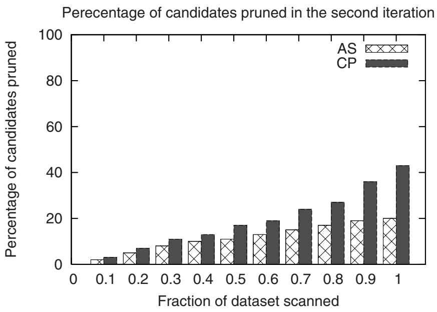
Item ordering. According to Equation 5, the initial value of a counter $d_p(X')$ is given by $d_p(X', 0) = S_e(X')$, i.e., the expected support of the prefix X'. Since candidates are pruned if a prefix decremental counter drops below the support requirement, it makes sense to pick those prefixes X' such that their initial values are as small as possible. A heuristic would be to set the item ordering Φ in increasing order of items' supports. We adopt this strategy for the CP method.

4 Experimental Evaluation

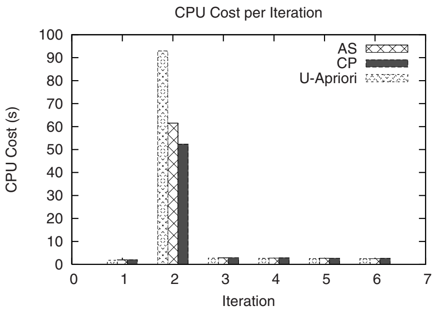
We conducted experiments comparing the performance of the DP methods against U-Apriori and LGS-Trimming. The experiments were conducted on a 2.6GHz P4 machine with 512MB memory running Linux Kernel 2.6.10. The algorithms were implemented using C.

We use the two-step dataset generation procedure documented in [4]. In the first step, the generator uses the IBM synthetic generator [1] to generate a dataset that contains frequent itemsets. We set the average number of items per transaction (T_{high}) to 20, the average length of frequent itemsets (I) to 6, and the number of transactions (D) to 100K[3]. In the second step, the generator uses an

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



(a)



(b)

Fig. 3. a) Percentage of candidates pruned in the 2nd iteration. b) CPU cost in each iteration.

uncertainty simulator to generate an existential probability for each item. The simulator first assigns each item in the dataset with a relatively high probability following a normal distribution with mean *HB* and standard deviation *HD*. To simulate items with low probabilities, the simulator inserts T_{low} items into each transaction. The probabilities of these items follow a normal distribution with mean *LB* and standard deviation *LD*. The average number of items per transaction, denoted by T, is equal to $T_{high}+T_{low}$. A parameter R is used to control the percentage of items with low probabilities in the dataset (i.e. $R = \frac{T_{low}}{T_{high}+T_{low}}$).

As an example, $T25/R20/I6/D100K/HB75/HD15/LB25/LD15$ represents an uncertain dataset with 25 items per transaction on average. Out of the 25 items, 20 are assigned with high probabilities and 5 are assigned with low probabilities. The high (low) probabilities are generated following a normal distribution with mean equal to 75% (25%) and standard deviation equal to 15% (15%). We call this dataset *Synthetic-1*.

4.1 Pruning Power of the Decremental Methods

In this section we investigate the pruning power of the decremental methods. The dataset we use is *Synthetic-1* and we set $\rho_s = 0.1\%$ in the experiment. Figure 3a shows the percentage of candidates pruned by AS and CP in the second iteration after a certain fraction of the dataset transactions have been processed. For example, the figure shows that about 20% of the candidates are pruned by CP after 60% of the transactions are processed. From the figure, we observe that the pruning power of CP is higher than that of AS. In particular, CP prunes twice as many candidates as AS after the entire dataset is scanned.

Recall that the idea of AS and CP is to replace a group of decremental counters by either a singleton decremental counter (AS-counter) or a prefix decremental counter (CP-counter). We say that an AS- or CP-counter $d_{s/p}(X')$ "covers" a decremental counter $\hat{S}_e(X, X')$ if $\hat{S}_e(X, X')$ is replaced by $d_{s/p}(X')$. Essentially, an AS- or CP-counter serves as an upper bound of a group of decremental counters covered by it. In the 2nd iteration, candidates are of size 2 and

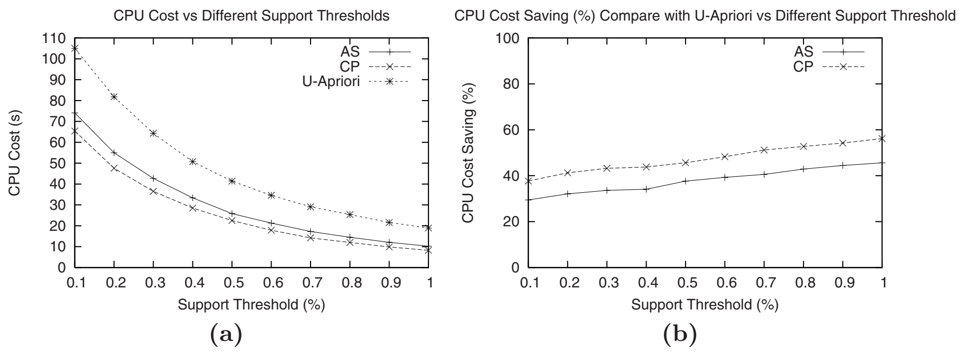


Fig. 4. CPU cost and saving with different ρ_s

therefore all proper prefixes contain only one item. We note that in general, a CP-counter, say $d_p(\{a\})$ covers fewer decremental counters than its AS counterpart, say $d_s(\{a\})$. This is because $d_p(\{a\})$ covers $\hat{S}_e(X, \{a\})$ only if $\{a\}$ is a prefix of X, while $d_s(\{a\})$ covers $\hat{S}_e(X, \{a\})$ only if $\{a\}$ is contained in X. Since prefix is a stronger requirement than containment, the set of counters covered by $d_p(\{a\})$ is always a subset of $d_s(\{a\})$. Therefore, each CP-counter "covers" fewer decremental counters than an AS-counter does. CP-counters are thus generally tighter upper bounds, leading to a more effective pruning.

Figure 3b shows the CPU cost in each iteration of the mining process. We see that in this experiment the costs of the 2^{nd} iteration dominates the others under all three algorithms. The pruning effectiveness of AS and CP in the 2^{nd} iteration (Figure 3a) thus reflects the CPU cost savings. For example, the 40% candidate reduction of CP translates into about 40s of CPU cost saving. Another observation is that although CP prunes twice as much as AS, the CPU cost saving of CP is not double of that of AS. This is because CP requires a more complex recursive strategy to maintain the prefix decremental counters, which is comparatively more costly.

4.2 Varying Minimum Support Threshold

Our next experiment compares the CPU costs of the DP methods against U-Apriori as the support threshold ρ_s varies from 0.1% to 1.0%. Figure 4a shows the CPU costs and Figure 4b shows the percentage of savings over U-Apriori. For example, when $\rho_s = 1\%$, CP saves about 59% of CPU time compared with U-Apriori. From the figures, we see that CP performs slightly better than AS over a wide range of ρ_s value. Also, the CPU costs of both CP and AS decrease as ρ_s increases. This is because a larger ρ_s implies fewer candidates and frequent itemsets, so the algorithms execute faster. Also, a larger ρ_s implies the minimum support requirement is larger. Hence, it is easier for the decremental counters to drop below the required value and more candidates can be pruned early.

4.3 Comparing with Data Trimming

Recall that LGS-Trimming consists of three steps: (1) remove low-probability items from dataset D to obtain a trimmed dataset D^T; (2) mine D^T; (3) patch up and recover missed frequent itemsets. LGS-Trimming and DP methods are orthogonal and can be combined. (DP can be applied to mining D^T and it also helps the patch-up step, which is essentially an additional iteration of candidate-generation and support-counting). In this section we compare U-Apriori, AS, CP, LGS-Trimming, and the combined method that integrates CP and LGS-Trimming. In particular, we study how the percentage of low-probability items (R) affects the algorithms' performance. In the experiment, we use *Synthetic-1* and set $\rho_s = 0.1\%$. Figure 5a shows the CPU costs and Figure 5b shows the percentage of savings over U-Apriori. From Figure 5b, we see that the performance of LGS-Trimming is very sensitive to R. Trimming outperforms AS and CP when R is large (e.g., 50%). This is because when there are numerous low-probability items, the trimmed dataset D^T is very small, and mining D^T is very efficient. On the other hand, if R is small, Trimming is less efficient than DP methods, and it could even be counter-productive for very small R. This is because for small R, D^T is large, so not much savings can be achieved by mining a trimmed dataset to compensate for the patch-up overhead.

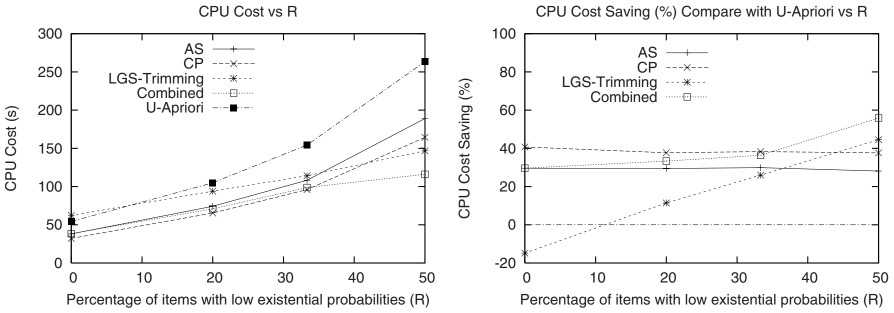


Fig. 5. CPU cost and saving with different R

In contrary, the performance of the DP methods are very stable over the range of R values. To understand this phenomenon let us consider Equation 4 for updating a AS-counter. The value of a AS-counter is determined by three terms: $S_e(x)$, $P_{t_k}(x)$ and $max_s(k)$. We note that varying R has small impact on the value of $S_e(x)$ because $S_e(x)$ is the expected support of item x, which is mainly determined by the high-probability entries of x in the dataset. Also, if transaction t_k contains a small-probability entry for x, then $P_{t_k}(x)$ is small and so the decrement to the value $d_s(x,k)$ would be insignificant. Hence, the population of small-probability items (i.e., R) has little effect in the decremental process. Finally, since $max_s(k)$ is determined by the maximum existential probability of the items (except x) in transaction t_k, low-probability items have little effect on the value of $max_s(k)$. As a result, the performance of AS is not sensitive to the population of low-probability items. A similar conclusion can be drawn for CP by considering Equation 5.

From the figures, we also observe that the combined algorithm strikes a good balance and gives consistently good performance. It's performance is comparable to those of AS and CP when R is small, and it gives the best performance when R is large.

5 Conclusions

In this paper we proposed a decremental pruning (DP) approach for efficient mining of frequent itemsets from existential uncertain data. Experimental results showed that DP achieved significant candidate reduction and computational cost savings. Compared with LGS-Trimming, DP had the advantages of not requiring a trimming threshold and its performance was relatively stable over a wide range of low-probability-item population. In particular, it outperformed data trimming when the dataset contained few low-probability items. We argued that the Trimming approach and the DP approach were orthogonal to each other. We showed that the two approaches could be combined leading to a generally best overall performance.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of 20th ICDE, pp. 487–499. Morgan Kaufmann, San Francisco (1994)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proc. of the 11th ICDE, pp. 3–14. IEEE Computer Society Press, Los Alamitos (1995)
3. Brossette, S.E., Sprague, A.P., Hardin, J.M., Jones, W.T., Moser, S.A.: Association rules and data mining in hospital infection control and public health surveillance. Journal of the American Medical Informatics Association, 373–381 (1998)
4. Chui, C.K., Kao, B., Hung, E.: Mining frequent itemsets from uncertain data. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 47–58. Springer, Heidelberg (2007)
5. Zimányi, E., Pirotte, A.: Imperfect information in relational databases. In: Uncertainty Management in Information Systems, pp. 35–88 (1996)
6. Bayardo Jr., R.J.: Efficiently mining long patterns from databases. In: Proc. of SIGMOD 1998, pp. 85–93. ACM Press, New York (1998)