An Efficient Polynomial Delay Algorithm for Pseudo Frequent Itemset Mining

Takeaki Uno[1], Hiroki Arimura[2]

[1] National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp
[2] Graduate School of Information Science and Technology, Hokkaido University
Kita 14 Nishi 9, Sapporo 060-0814, Japan
arim@ist.hokudai.ac.jp

Abstract. Mining frequently appearing patterns in a database is a basic problem in informatics, especially in data mining. Particularly, when the input database is a collection of subsets of an itemset, the problem is called the frequent itemset mining problem, and has been extensively studied. In the real-world use, one of difficulties of frequent itemset mining is that real-world data is often incorrect, or missing some parts. It causes that some records which should include a pattern do not have it. To deal with real-world problems, one can use an ambiguous inclusion relation and find patterns which are mostly included in many records. However, computational difficulty have prevented such problems from being actively used in practice. In this paper, we use an alternative inclusion relation in which we consider an itemset P to be included in an itemset T if at most k items of P are not included in T, i.e., $|P \setminus T| \leq k$. We address the problem of enumerating frequent itemsets under this inclusion relation and propose an efficient polynomial delay polynomial space algorithm. Moreover, To enable us to skip many small non-valuable frequent itemsets, we propose an algorithm for directly enumerating frequent itemsets of a certain size.

1 Introduction

The frequent pattern mining problem is to find patterns frequently appearing in a given database. It is one of the central tasks in data mining, and has been a focus of recent informatics studies. Particularly, when the database is a collection of transactions and the patterns to be found are also subsets of itemsets, the problem is called the frequent itemset mining problem[1, 4, 10–12]. Precisely, we define the *frequency* of an itemset by the number of transactions including the pattern, and say an itemset is a *frequent itemset* if its frequency is no less than the given threshold value σ, called *minimum support.*

Frequent pattern mining is often used especially for data analysis. For data so huge that humans can not get any intuition from an overview of it, the frequent pattern mining is a useful way to capture the features of the data's features, both in a global sense and in a local sense. However, in the real world use, we often encounter difficulties in trying to use the frequent pattern mining on real-world data. One difficulty is that data are often incorrect or missing parts. Such errors mean that some records that should include a pattern P do not include P, thus P may be overlooked because its frequency appears to be too low. A way to deal with this difficulty is to consider an ambiguous inclusion relation whereby we consider that a transaction T includes a pattern P if most items of P are included in T.

There are several studies on the frequent pattern mining with ambiguous inclusions. In some contexts, these patterns are called fault-tolerant frequent itemsets[5–8, 14]. In some of these studies, ambiguous inclusion is defined such that an itemset P is included in a transaction T if the fraction of items of P included in T is no less than a given threshold θ, i.e., $|P \cap T|/|P| \geq \theta$[14]. Given this definition, the family of frequent itemsets is not always anti-monotone; thus the usual apriori based algorithms are not output sensitive in the sense of time complexity.

If an item of the itemset is not included in a transaction, then it can be considered to be a fault. Some studies, such as Boulicaut et al., Liu et al., and Seppanen et al.[5–8], treat mining pairs of an itemset and a transaction set such that there are few faults between their elements. When the size of the transaction set is large, we can regard the itemset as a frequent pattern with ambiguous inclusions. Many mining algorithms have been devised for solving both problems, but enumeration difficulties prevent them from having the completeness that ensures that they output exactly all frequent patterns.

On the other hand, in sequence pattern mining and text mining, ambiguous matching is used to define the occurrence of a pattern, i.e., if a pattern is homogeneous to a substring of the input string, then we regard that the pattern appears at the position. Such patterns are called degenerate patterns in some contexts, especially in genome sciences, and several algorithms have been proposed[9, 13].

There are possibly several models for such ambiguous inclusions. In this paper, we define our ambiguous relation with a constant k that a pattern P is included in a transaction (or pattern) T if at most k items are not included in T. In this paper, we address the problem of enumerating all frequent itemsets under this inclusion relation, for given a transaction database, minimum support σ, and k. When σ is large, such as 90% of the number of transactions, the problem can be considered to be one to find combinations of items $i_1, \ldots, i_h$ such that at least $h - k$ items are included in 90% of transactions, thus such combinations characterize the database. These combinations can also be used as rules separating the database from other database, thus has applications to learning theory and practice.

For the frequent itemset mining with our ambiguous relation, we propose a polynomial delay polynomial space algorithm. To best of our knowledge, this is the first result of even output polynomial time algorithm for this problem. Although the algorithm is polynomial time, we still encounter a problem in the real-world applications, that is, quite many uninteresting small patterns are frequent in our ambiguous inclusion relation. We can avoid this problem by directly enumerating all frequent patterns of given size l, and we propose an efficient algorithm for this task.

The organization of the paper is as follows. We introduce several notations and notions concerning to our ambiguous inclusion in Section 2, and propose a basic algorithm for frequent itemset mining under the condition of he ambiguous inclusion in Section 3. The algorithm is improved in Section 4. Section 5 describes an algorithm for directly enumerating those of size l, and Section 6 is for the conclusion.

2 Preliminary

Let an *itemset* I be a set of items $1, \dots n$. A *transaction database* $\mathcal{D}$ is a collection of transactions where a *transaction* is a subset of I[3]. We denote the number of transactions in $\mathcal{D}$ by $|\mathcal{D}|$, and the size of $\mathcal{D}$ by $||\mathcal{D}||$. Here the size of $\mathcal{D}$ is the sum of $|\mathcal{D}|$ and the sizes of the transactions in $\mathcal{D}$, i.e., $||\mathcal{D}|| = |\mathcal{D}| + \sum_{T \in \mathcal{D}} |T|$. Note that $||\mathcal{D}||$ is not defined in the usual sense. The aim of this definition is to consider the computation time for empty transactions, which is $O(1)$. Hereafter, we fix the database $\mathcal{D}$ for the input of the algorithm.

A *pattern* P is a subset of itemset I. The largest item in P is called the *tail* of P and is denoted by $tail(P)$. A transaction of $\mathcal{D}$ including P is called an *occurrence* of P. We denote the set of occurrences of P by $Occ(P)$. The *frequency* $frq(P)$ of a pattern P is defined by the number of transactions including P, i.e., $|Occ(P)|$. Given a transaction database $\mathcal{D}$ and constant number σ, a pattern with frequency no less than σ is called a *frequent itemset*. The frequency is often called *support*, and σ is called the *minimum support*. The problem of finding all frequent itemsets for given a transaction database and minimum support is called the *frequent itemset enumeration problem*[4].

For a constant k and two patterns $P, T \subseteq I$, we write $P \subseteq_k T$ if $|P \setminus T| \leq k$ holds. We call the binary relation $\subseteq_k$ the *k-pseudo inclusion relation*. For a pattern P, a transaction T is a *k-pseudo occurrence* of P if $P \subseteq_k T$. We denote the set of k-pseudo occurrences of P by $Occ_{\leq k}(P)$. Particularly, the set of transactions satisfying $|P \setminus T| = k$ is denoted by $Occ_{=k}(P)$. We have $Occ(P), Occ_{=k}(P) \subseteq Occ_{\leq k}(P)$. See the example in Fig. 2. We define the *k-pseudo frequency* of P by $|Occ_{\leq k}(P)|$, and denote it by $frq_k(P)$. A *k-pseudo frequent itemset* is a pattern P such that its k-pseudo frequency is no less than σ. Here, we define our problem as follows.

[3] In the literatures, a transaction is often defined by a pair of an item subset and its ID. However, we will here omit the ID since ID has no meaning in the arguments in this paper.

[4] This problem is also called frequent itemset/pattern mining/discovery. Usually, the terms mining and discovery do not require the output to be complete, thus here we use the term enumeration which is used in the problem of outputting all the solutions completely.

A: 1,2,4,5,6
B: 2,3,4
C: 1,2,7
D: 1,5
E: 2,3,7
F: 2,7
G: 4
H: 6

$Occ_{\le 0}(\{2,7\}) = \{E,F\}$
$Occ_{\le 1}(\{1,2,4\}) = \{A,B,C\}$
$Occ_{=1}(\{1,3,7\}) = \{C,E\}$

$Occ_{\le 2}(\{1,2,4,7\}) = \{A,B,C,E,F\}$
$Occ_{=2}(\{1,2,4,7\}) = \{B,C,E,F\}$
$Occ_{\le 2}(\{1,2,4,7\} \cup \{3\})$
$= Occ_{\le 1}(\{1,2,4,7\}) \cup (Occ_{=2}(\{1,2,4,7\}) \cap Occ(\{3\}))$
$= \{A,E\}$

Fig. 1. Examples of pseudo occurrences and update by addition of an item

Pseudo Frequent Itemset Enumeration Problem
Input: transaction database $\mathcal{D}$, minimum support σ, constant k
Output: all k-pseudo frequent itemsets in $\mathcal{D}$

If an algorithm terminates in polynomial time for both the input size and the output size, the algorithm is called *output polynomial.* Output polynomiality is a popular measure of the theoretical efficiency of the algorithm. If the computation time between any two consecutive output solutions is bounded by a polynomial of the input size, the algorithm is called *polynomial delay.* If an algorithm is polynomial delay, the computation time is linear in the number of outputs, and hence better in practice. If the memory usage of the algorithm is bounded by a polynomial of the input size, the algorithm is called *polynomial space.* Here our goal is to develop an efficient polynomial delay polynomial space algorithm for solving the pseudo frequent itemset enumeration problem.

3 Basic Algorithm

The frequent itemset enumeration problem is, from the viewpoint of complexity theory, an easy problem. The reason is that the frequency has a monotone property, thus obviously any frequent itemset can be obtained by iteratively adding items to the emptyset by passing through only frequent itemsets. Although a naive implementation may produce duplicate solutions, we can avoid duplications by using tail extension. For a frequent itemset P, a pattern obtained from P by adding an item larger than the tail of P is called a *tail extension* of P. By generating frequent patterns only via tail extensions, each pattern is generated only from the pattern obtained by removing its tail, thus we can enumerate all frequent itemsets without duplicates. A backtrack algorithm generates tail extensions in a depth-first manner, and thus is a polynomial time delay polynomial space algorithm. Precisely, the computation time for each frequent itemset is linear in the size of the database, i.e., $O(||\mathcal{D}||)$. The space complexity is also optimal, that is, $O(||\mathcal{D}||)$.

Regarding the practical use of frequent itemset enumeration, the number of frequent itemsets is usually not so large compared to the input size, but the input size is usually large. Thus, linear time in the input size for each solution

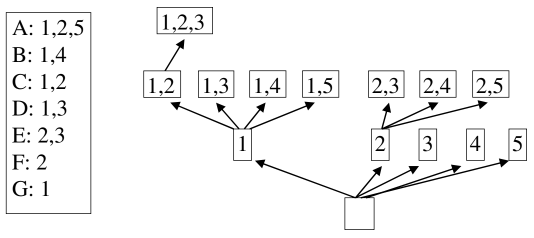

Fig. 2. An example of backtrack algorithm execution for minimum support $\sigma = 4$

is too long. To reduce the practical computation time, several techniques have been proposed. One of the most efficient techniques is called database reduction.

Consider the following operation: Remove all items included in less than σ transactions, and unify the same transactions into one transaction. Then, the database shrinks, and its size becomes small. We can further reduce the size by using trie or prefix tree. This operation is called *database reduction*. Database reduction performs well in practice, especially when σ is large, Moreover, if we apply database reduction to *conditional databases* to recursively reduce their sizes, we can further reduce the computation time. Here a conditional database is the database restricted to items larger than the tail of the current operating pattern P and transactions including P, which is the input of an iteration with respect to P. This technique is called *iterated database reduction*.

These techniques can be applied to pseudo frequent itemsets in a similar way. We begin with the following proposition to see the monotonicity.

Proposition 1. *For any patterns P and P' satisfying $P \subseteq P'$, $Occ_{\leq k}(P') \subseteq Occ_{\leq k}(P)$ holds.*

The statement holds since any transaction $T \in Occ_{\leq k}(P')$ does not include at most k items in P. From this proposition, we can see that the family of k-pseudo frequent itemsets satisfies anti-monotonicity. Hereafter, we assume that the minimum support is from 1 to $|\mathcal{D}|$ thus the emptyset is always k-pseudo frequent. For a k-pseudo frequent itemset P, let the children set of P, denoted by CHD(P), be the set of items i such that $i > tail(P)$ and $P \cup \{i\}$ is k-pseudo frequent. The monotone property leads to the following backtrack algorithm. By calling **Backtrack**$(\emptyset)$, we can enumerate all k-pseudo frequent itemsets.

```
backtrack($P$)
1. Output $P$
2. Compute CHD($P$)
3. for each $i \in$ CHD($P$) call backtrack ($P \cup \{i\}$)
```

It is easy to see the correctness of this algorithm. Figure 3 shows an execution of the backtrack algorithm. Each iteration inputs a k-pseudo frequent itemset P, outputs P, computes the k-pseudo frequency for all tail extensions of P to obtain CHD(P), and generates recursive calls for each item in CHD(P). Thus, any iteration outputs a k-pseudo frequent itemset, and the computation time for each k-pseudo frequent itemset is bounded by the maximum computation time of an iteration. Computing k-pseudo frequency of each tail extension takes $O(||\mathcal{D}||)$ time thus the computation time of an iteration is $O(n||\mathcal{D}||)$. This can be shortened as follows.

Suppose that we have $Occ_{=0}(P), ..., Occ_{=k}(P)$ for the current processing pattern P. Here we consider the computation of $Occ_{=0}(P \cup \{i\}), ..., Occ_{=k}(P \cup \{i\})$ for all $i > tail(P)$. First, we prove the following proposition.

Proposition 2. *For a transaction T included in $Occ_{=h}(P)$ for some $h, 0 \leq h \leq k$, $T \in Occ_{=h}(P \cup \{i\})$ holds if T includes i. Otherwise, $T \in Occ_{=h+1}(P \cup \{i\})$.*

Proof. Since $T \in Occ_{=h}(P)$, T does not include exactly h items of $(P \cup \{i\})$ if T includes i, and exactly h items otherwise. Then the statement follows. $\square$

Now we have the following lemma.

Lemma 1. *The following two equations hold,*
(a) $Occ_{=0}(P \cup \{i\}) = Occ_{=0}(P) \cap Occ(\{i\})$
(b) $Occ_{=h}(P \cup \{i\}) = (Occ_{=h}(P) \cap Occ(\{i\})) \cup (Occ_{=h-1}(P) \setminus Occ(\{i\}))$ for any $h \geq 1$.

Proof. Any transaction $T \in Occ_{=h}(P \cup \{i\}), 0 \leq h \leq k$, includes at least $h-1$ and at most h items of P. This implies that T is included in $Occ_{=h}(P)$ or $Occ_{=h-1}(P)$ only when $h > 0$. On the other hand, from Proposition 2, we have

$$Occ_{=h}(P \cup \{i\}) \cap Occ_{=h}(P) = Occ_{=h}(P) \cap Occ(\{i\}), \text{and}$$

$$Occ_{=h}(P \cup \{i\}) \cap Occ_{=h-1}(P) = Occ_{=h-1}(P) \setminus Occ(\{i\}), \text{ for } h > 0.$$

Thus, the statement of the lemma holds. $\square$

The next proposition is a consequence of the lemma.

Proposition 3. *$Occ_{\leq k}(P \cup \{i\}) = Occ_{\leq k-1}(P) \cup (Occ_{=k}(P) \cap Occ(\{i\}))$.*

From Lemma 1, we can see that all we have to do is take intersection of occurrences for all i. For this task, the technique so called *occurrence deliver* described in [10, 12, 11] is efficient.

Let us consider the task of computing $Occ_{=k}(P \cup \{i\})$ for all $i > tail(P)$. First, we prepare an empty bucket for each item i. Next, for each transaction T in $Occ_{=k}(P)$, we do "insert T into the bucket of i for each item $i \in T, i > tail(P)$". After performing this operation for all transactions in $Occ_{=k}(P)$, the content of the bucket of i is equal to $Occ_{=k}(P \cup \{i\})$. The pseudo code of occurrence deliver is described as follows. The code inputs a set of transactions $\mathcal{S}$ and pattern P, then sets $bucket[i]$ to $\mathcal{S} \cap Occ(\{i\})$ for all $i > tail(P)$. We suppose that the bucket of any item i is initialized, and thus is empty at the beginning.

```
**Occurrence deliver**($\mathcal{S}$, P)
1. **for each** $T \in \mathcal{S}$ **do**
2.     **for each** $i \in T, i > tail(P)$ **do**
```

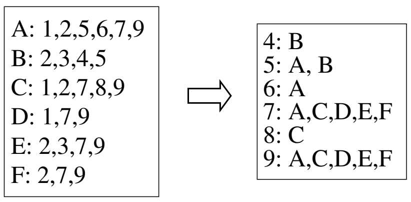

Fig. 3. Example execution of occurrence deliver

```
3.        insert $T$ into $bucket[i]$
4.    end for
5. end for
```

Fig. 3 shows an example of the execution of occurrence deliver. Let $\mathcal{S}_{>h} = \{T \cap \{h+1, \ldots, |I|\} \mid T \in \mathcal{S}\})$. Hereafter, we assume that each transaction T is stored in memory so that the items in T are sorted in increasing order of items. Bucket sort or radix sort to all transactions at once can be done in $O(||\mathcal{D}|| + |I|)$ time. The following proposition is proved in [10–12].

Lemma 2. *Algorithm* **Occurrence deliver** *takes $O(||\mathcal{S}_{>tail(P)}||)$ time and computes $\mathcal{S} \cap Occ(\{i\})$ for all $i > tail(P)$.*

Lemma 2 leads in turn to the following proposition.

Proposition 4. *For pattern P, we can compute the k-pseudo frequency of all $P \cup \{i\}, i > tail(P)$ having non-zero k-pseudo frequency in $O(||Occ_{=k}(P)_{>tail(P)}||)$ time.*

From Proposition 4, we can see that computation of CHD(P) can be done in $O(||Occ_{=k}(P)_{>tail(P)}||) = O(||\mathcal{D}||)$ time. Next let us consider the cost of computing $Occ_{=0}(P \cup \{i\}), ..., Occ_{=k}(P \cup \{i\})$ for each $i \in$ CHD(P). From Lemma 1, we can see that it can be computed by taking the intersection of $Occ_{\leq k}(P)$ and $Occ(\{i\})$ in $O(|Occ_{\leq k}(P)| + |Occ(\{i\})|)$ time. The following proposition is stated for the memory use[10, 12, 11].

Proposition 5. *For any set $\mathcal{S} \subseteq \mathcal{D}$ of transactions and item i, the size of the bucket of i does not exceed $|Occ(\{i\})|$ after applying occurrence deliver.*

We can see from Proposition 5 that the memory used by an iteration is bounded by $O(||\mathcal{D}||)$. The depth of the recursion of **Backtrack** is at most n, and the accumulated memory usage is $O(n||\mathcal{D}||)$.

Theorem 1. *For given a database $\mathcal{D}$, minimum support σ and constant k, algorithm* **Backtrack** *enumerates k-pseudo frequent itemsets in $O(N \cdot ||\mathcal{D}||)$ time with using $O(n||\mathcal{D}||)$ memory, where N is the number of k-pseudo frequent itemsets.*

Corollary 1. *Algorithm* **Backtrack** *is a polynomial delay polynomial space algorithm for enumerating all k-pseudo frequent itemsets.*

4 Reducing Computational Cost

In this section, we improve the efficiency of the algorithm proposed in the previous section by reducing both time and space complexities. Our basic idea is to re-use one bucket in all iterations. This results in a reduction of memory usage.

Here we denote a transaction T in $Occ_{=h}(P \cup \{i\})$ by a pair (T, h). Instead of having all $Occ_{=0}(P \cup \{i\}), ..., Occ_{=k}(P \cup \{i\})$, we maintain $Occ'_{\leq k}(P) = \{(T, h) \mid T \in Occ_{=h}(P \cup \{i\})\}$ keeping that all elements (T, h) in $Occ'_{\leq k}(P)$ are sorted in increasing order of h. Then, by applying occurrence deliver to $Occ'_{\leq k}(P)$, we can obtain $Occ'_{\leq k}(P \cup \{i\})$ while keeping the order in $O(||Occ_{=k}(P)_{>tail(P)}||)$ time. By looking at the bottom of each bucket, we can easily take $Occ_{=k}(P)$ in $O(|Occ_{=k}(P)|)$ time. This simplifies the operation to maintain the $Occ_{=h}$ for all h.

A technique called *rightmost sweep* is useful for the re-use of buckets[10]. The following propositions and lemmas regard the availability of buckets.

Proposition 6. *For an iteration inputting pattern P, no bucket of $i \leq tail(P)$ is accessed from the beginning of the iteration to the termination of the iteration, including the execution of the recursive calls.*

An iteration adds items i larger than $tail(P)$, and $tail(P \cup \{i\}) > tail(P)$ always holds. Occurrence deliver accesses only the buckets of i satisfying $i > tail(P)$; thus the statement holds. Proposition 6 indicates that when we generate a recursive call with respect to $P \cup \{i\}$, the bucket of any $j < i$ is preserved until the end of the recursive call. Thus, we consider the following algorithm PFIM (Pseudo Frequent Itemset minor) that generates recursive calls in decreasing order of indices.

```
PFIM($P$, $Occ'_{\leq k}(P)$)
1. Output $P$
2. Apply occurrence deliver to $Occ'_{\leq k}(P)$
3. if $|Occ_{\leq k-1}(P)| \geq \sigma$ then $L := \{tail(P) + 1, \ldots, n\}$
4. else $L := \{i \mid |Occ_{=k}(P \cup \{i\})| > 0\}$
  remove $i \notin$ CHD($P$) from $L$ and initialize the bucket of $i$
5. end if
6. sort items in $L$ in the decreasing order
7. while $L \neq \emptyset$ do
8.    extract the head $i$ of $L$
9.    call PFIM ($P \cup \{i\}, Occ'_{\leq k}(P \cup \{i\})$)
10.   initialize the bucket of $i$
11. end while
```

This algorithm re-uses buckets; thus the buckets to be used seem to be not initialized at the beginning of an iteration. However, if we can prove that those buckets are actually initialized at the beginning, we can be assured of the correctness of the algorithm.

Lemma 3. *If all buckets of $i > tail(P)$ are initialized at the beginning of an iteration of* **PFIM** *inputting pattern P, then the buckets of $i > tail(P)$ are also initialized at the termination of the iteration.*

Proof. We prove the statement by the induction, starting from the leaves of the computation tree of the algorithm. For any iteration, we define its height by 0 if it generates no recursive call, and the maximum height plus one otherwise. The height is the distance to the farthest leaf among its descendants in the computation tree.

First, we consider an iteration that generates no recursive call. In such an iteration, all buckets inserted some elements in step 2 are initialized in step 4, thereby L has no element. Thus, the statement holds.

Next, we suppose that for any iteration of height at most h satisfies the statement, and we consider an iteration I of height $h + 1$. Let P be the input pattern of I and suppose that at the beginning of I, the bucket of any $i > tail(P)$ is initialized. Of the buckets holding some elements in step2, the buckets of $i \notin \mathrm{CHD}(P)$ are initialized in step 4. Several recursive calls are generated in the loop from step 7 to step 12. Suppose that i is the head of L. When we generate the recursive call with $P \cup \{i\}$, the bucket of any $j > tail(P \cup \{i_1\})$ is initialized since L is sorted in decreasing order. From the assumption of the induction, the bucket of any $j > tail(P \cup \{i\})$ is initialized after the termination of the recursive call. Then, the bucket of i is initialized. Since i is extracted from L, for the new head i' of L, the bucket of any $j > i'$ is again initialized. In this way, recursive calls are generated with satisfying the assumption of the statement. Thus, after generating recursive calls for all items in L, the bucket of any $j > tail(P)$ is initialized. □

Form the lemma, we obtain the following theorem.

Theorem 2. *Algorithm* **PFIM** *uses $O(||\mathcal{D}||)$ memory and enumerates all k-pseudo frequent itemsets in $\mathcal{D}$ in $O(\sum_{P\in\mathcal{F}} ||Occ_{\leq k}(P)_{>tail(P)}|| + \log n) = O(|\mathcal{F}| \times ||\mathcal{D}||)$ time, where $\mathcal{F}$ is the family of k-pseudo frequent itemsets.*

Proof. The correctness of the algorithm is obvious from the correctness of Algorithm **Backtrack** and Lemma 3. The statement for the memory usage is clear from the re-use of buckets.

Next, we discuss the computation time. Step 2 is done in $O(||Occ_{\leq k}(P)_{>tail(P)}||)$ time, and step 6 is done in $O(|\mathrm{CHD}(P)| \log n)$ time. Other steps can be done in $O(|\mathrm{CHD}(P)|)$ time. Thus, by taking the sum over all k-pseudo frequent itemsets, the total computation time is bounded by $O(\sum_{P\in\mathcal{F}}(||Occ_{\leq k}(P)_{>tail(P)}|| + \log n)) = O(|\mathcal{F}| \times ||\mathcal{D}||)$. □

The structure of the algorithms is almost equal to that of LCM[10–12] for the frequent itemset enumeration. Our algorithm can be used together with practical efficient techniques such as database reduction, thus our algorithm should perform well in practice.

5 Efficient Computation in Practice

In this paper, we use the k-pseudo inclusion relation as a model of ambiguous inclusion. Although this is a natural modeling, it has a weak point in practice; that is, many small patterns are k-pseudo frequent. For example, any pattern whose size is no greater than k is a k-pseudo frequent itemset, and an addition of any item to a $(k-1)$-pseudo frequent itemset also yields a k-pseudo frequent itemset. In the real-world problems, we may not have much interest in these small patterns.

To cope with this difficulty, we often enumerate only the maximal patterns in the sense of set inclusion. However, possibly so many small itemsets have k-pseudo frequencies close to the minimum support, many of these small patterns become maximal. Moreover, we lose non-maximal but large k-pseudo frequent itemsets. Thus, we here address the method for enumerating k-pseudo frequent itemsets of given size l directly. For a pattern P and its item i, let $Occ^*_{=k}(P,i)$ be the set of k-pseudo occurrences T of P such that T does not include i, i.e., $Occ^*_{=k}(P,i) = \{T \mid T \in Occ_{=k}(P), i \notin T\}$.

Lemma 4. *For any pattern P, there exists a sequence of its items $(i_1, i_2, \ldots, i_{|P|})$ such that for any y, $|Occ_{\le k-1}(\{i_1,\ldots,i_y\})| \ge |Occ_{\le k}(P)|\frac{|P|-y}{|P|}$ holds.*

Proof. Let $(i_1, i_2, \ldots, i_{|P|})$ be the items of P sorted in increasing order of $|Occ^*_{=k}(P,i_j)|$, i.e., for any $1 \le y < |P|$, $|Occ^*_{=k}(P,i_y)| \le |Occ^*_{=k}(P,i_{y+1})|$ holds. Consider the $(k-1)$-pseudo frequency of pattern $\{1,\ldots,y\}$ for some $1 \le y < |P|$. For any $j > y$, $\{1,\ldots,y\}$ is included in any transaction of $Occ^*_{=k}(P,i_j)$ in the sense of $(k-1)$-pseudo inclusion. Observe that the average of $|Occ^*_{=k}(P,i_j)|, 1 \le j \le |P|$ is at most $|Occ_{=k}(P)|\frac{k}{|P|}$, and one transaction is included in $Occ^*_{=k}(P,i_j)$ at most k j's. Thus, we see that the cardinality of $\bigcup_{j=y+1}^{|P|} Occ^*_{=k}(P,i_j)$ is at least $(|P|-y) \times |Occ_{=k}(P)|\frac{k}{|P|}/k = |Occ_{=k}(P)|\frac{|P|-y}{|P|}$. Since $|Occ_{\le k-1}(\{i_1,\ldots,i_y\})| = |Occ_{\le k-1}(P)| + |\bigcup_{j=y+1}^{|P|} Occ^*_{=k}(P,i_j)|$, the sequence $(i_1,\ldots,i_{|P|})$ satisfies the statement. □

For given a constant l and a pattern P such that $|P| < l$, we call the condition $|Occ_{\le k-1}(P)| \ge \sigma\frac{l-|P|}{l}$ the *partial frequency condition*, and we denote by $\mathcal{K}$ the set of all k-pseudo frequent itemsets of size less than l satisfying the partial frequency condition. From the lemma, we can see that any k-pseudo frequent itemset of size l can be generated by adding items by passing through only patterns in $\mathcal{K}$, thus we can use the condition for pruning the iterations. The size of $\mathcal{K}$ is expected to be smaller than that of k-pseudo frequent itemsets of sizes of at most l, thus the computation time will be short.

For such a generation, we can not use the usual tail extension, since for some $P \in \mathcal{K}$, $P \setminus \{tail(P)\}$ may not be in $\mathcal{K}$. On the other hand, if we add items smaller than the tail, we may produce a pattern $P = \{i_1,\ldots,i_h\} \in \mathcal{K}$ twice from $P \setminus \{i_j\}$, and $P \setminus \{i_g\}$ for some $j \neq g$, Thus, we consider the following generation rule to avoid duplicates.

Generation Rule: Generate each pattern $P \in \mathcal{K}$ only from the pattern $P \setminus \{i\}, i \in P$ maximizing $|Occ_{k-1}(P \setminus \{i\})|$ among all patterns obtained by removing an item from P. Ties are broken by lexicographical order.

Lemma 5. *Adding items under the generation rule, any $P \in \mathcal{K}$ is generated exactly once.*

An enumeration algorithm using such a generation rule is called *reverse search*[3]. The algorithm is as follows.

```
ReverseSearch ($P$)
1. if $|P| = l$ then output $P$ ; return
2. for each $i \notin P$ do
3.   if $|Occ_{\le k}(P \cup \{i\})| \ge \sigma$ then // $k$-pseudo frequency check
4.     if $|Occ_{\le k-1}(P \cup \{i\})| \ge \frac{\sigma}{l}(l - |P|)$ then // partial frequency check
5.       if $P$ and $P \cup \{i\}$ satisfy the generation rule then
         call ReverseSearch ($P \cup \{i\}$)
6. end for
```

Lemma 6. *The computation time of an iteration of the algorithm* **ReverseSearch** *is $O(|P| \times ||D||)$.*

Proof. The key to the computation time is steps 3, 4 and 5. For steps 3 and 4, we explained that they can be done in $O(||D||)$ time, thus we have to consider only step 5. It checks the generation rule, by computing $|Occ_{\le k-1}(P \cup \{i\} \setminus \{j\})|$ for all $j \in P$. This takes $O(||D|| \times |P|)$ in a straightforward way, we thereby explain how to decrease it.

Observe that $|Occ_{\le k-1}(P \cup \{i\} \setminus \{j\})| = |Occ_{\le k-1}(P \cup \{i\})| + |Occ{=}\ k(P \cup \{i\}) \setminus Occ(\{j\})|$. Since $Occ_{\le k-1}(P \cup \{i\})$ can be obtained in $O(||\mathcal{D}||)$ time, all we have to do is to compute $|Occ_{=k}(P \cup \{i\}) \setminus Occ(\{j\})|$ quickly. For the task, we maintain the set $Occ_{=k}(P) \cap Occ(\{j\})$ for $j \in P$ in memory, and update them in each iteration. This takes $O(||D||)$ time by occurrence deliver. Using these, we can compute $|Occ_{=k}(P \cup \{i\}) \setminus Occ(\{j\})|$ for all $j \in P$ in $O(||Occ_{=k}(P \cup \{i\})|| \times |P|)$ time. Since the sum of $||Occ_{=k}(P \cup \{i\})||$ over all $i \notin P$ never exceed $||D||$, the time to compute $|Occ_{=k}(P \cup \{i\}) \setminus Occ(\{j\})|$ for all pairs of i and j is $O(|P| \times ||D||)$. □

6 Conclusion and future work

In this paper, we introduced an ambiguous inclusion relation to the frequent itemset mining as a meaning of dealing with errors and ambiguities. We chose a model for ambiguous inclusion by relaxing the inclusion relation so that several items can be excluded, and formulated the pseudo frequent itemset enumeration problem by the inclusion relation. To solve the problem, we proposed an efficient polynomial delay polynomial space algorithm. The algorithm inherits the structure from the existing efficient frequent itemset mining algorithms, thus we expect that it will have high performance in practical use. To skip many small and non-valuable frequent itemsets, we propose an algorithm for directly enumerating frequent itemsets of a certain size. As future works, to evaluate the efficiency in the real-world problems implementation of the algorithm and computational experiments are crucial. Another interesting research topic is extensions of the technique in this paper to other frequent pattern mining problems.

Acknowledgments

Part of this research was supported by Grant-in-Aid for Scientific Research of Japan "Developing efficient and accurate algorithms for large-scale data processing in genome science", and and joint-research funds of the National Institute of Informatics.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo, *Fast Discovery of Association Rules*, In *Advances in Knowledge Discovery and Data Mining*, pp. 307–328, 1996.
2. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data. *SDM 2002*, 2002.
3. D. Avis and K. Fukuda, Reverse Search for Enumeration, *Discrete App. Math.,* 65, pp. 21–46, 1996.
4. R. J. Bayardo Jr., *Efficiently Mining Long Patterns from Databases*, In *Proc. SIGMOD'98*, pp. 85–93, 1998.
5. J. Besson, C. Robardet, and J. F. Boulicaut, Mining Formal Concepts with a Bounded Number of Exceptions from Transactional Data, *KDID 2004, Lecture Notes in Computer Science* 3377, pp. 33–45, 2005.
6. J. Liu, S. Paulsen, W. Wang, A. Nobel, J. Prins, "Mining Approximate Frequent Itemsets from Noisy Data," 5th IEEE International Conference on Data Mining (ICDM'05), pp. 721-724, 2005
7. J. K. Seppanen and H. Mannila, "Dense Itemsets", In SIGKDD 2004.
8. W. Shen-Shung and L. Suh-Yin, "Mining Fault-Tolerant Frequent Patterns in Large Databases", ICS2002, 2002.
9. M. Takeda, S. Inenaga, H. Bannai, A. Shinohara, and S. Arikawa, Discovering Most Classificatory Patterns for Very Expressive Pattern Classes, *In Proc. of Discovery Science 2003, Lecture Notes in Computer Science* 2843, pp. 486–493, 2003.
10. T. Uno, T. Asai, Y. Uchida, H. Arimura, LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets, In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003.
11. T. Uno, T. Asai, Y. Uchida, H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, *Lecture Notes in Artificial Intelligence* 3245, pp. 16–31, 2004.
12. T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets, In *Proc. IEEE ICDM'04 Workshop FIMI'04*, 2004.
13. J. T. L. Wang, G. W. Chirn, T. G. Marr, B. Shapiro, D. Shasha and K. Zhang, Combinatorial pattern discovery for scientific data: some preliminary results, *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pp. 115–125, 1994
14. C. Yang, U. Fayyad, P. S. Bradley, "Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions," In SIGKDD 2001.