# Ambiguous Frequent Itemset Mining and Polynomial Delay Enumeration

Takeaki Uno[1] and 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 recent informatics, especially in data mining. Particularly, when the input database is a collection of subsets of an itemset, called transaction, the problem is called the frequent itemset mining problem, and it has been extensively studied. The items in a frequent itemset appear in many records simultaneously, thus they can be considered to be a cluster with respect to these records. However, in this sense, the condition that every item appears in each record is quite strong. We should allow for several missing items in these records. In this paper, we approach this problem from the algorithm theory, and consider the model that can be solved efficiently and possibly valuable in practice. We introduce ambiguous frequent itemsets which allow missing items in their occurrence records. More precisely, for given thresholds $\theta$ and $\sigma$, an ambiguous frequent itemset $P$ has a transaction set $\mathcal{T}$, $|\mathcal{T}| \geq \sigma$, such that on average, transactions in $\mathcal{T}$ include ratio $\theta$ of items of $P$. We formulate the problem of enumerating ambiguous frequent itemsets, and propose an efficient polynomial delay polynomial space algorithm. The practical performance is evaluated by computational experiments. Our algorithm can be naturally extended to the weighted version of the problem. The weighted version is a natural extension of the ordinary frequent itemset to weighted transaction databases, and is equivalent to finding submatrices with large average weights in their cells. An implementation is available at the author's homepage[3]

## 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 $\mathcal{D}$ is a collection of *transactions*[4] where a transaction is a subset of an *itemset* $I = \{1, \ldots n\}$, and the *patterns* to be found are also subsets of itemsets, the problem is called the *frequent itemset mining problem*[1, 4, 11, 12].

[3] http://research.nii.ac.jp/~uno/index.html
[4] In the literature, a transaction is often defined by a pair of an item subset and its ID. However, we omit the ID since it has no effect on the arguments in this paper.

Precisely, a transaction of $\mathcal{D}$ including $P$ is called an *occurrence* of $P$, and the set of occurrences of $P$ is denoted by $Occ(P)$. we define the *frequency* of an itemset by $|Occ(P)|$, and say an itemset is a *frequent itemset* if its frequency is no less than the given threshold value $\sigma$, called *minimum support.* The frequency is often called *support*, and $\sigma$ is called the *minimum support.*

Frequent pattern mining is often used 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, we often encounter difficulties in trying to use the frequent pattern mining on real-world data. One difficulty is that data is often incorrect or has missing parts. Such errors mean that some records that should include a pattern $P$ do not include it, 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, 7–9, 16]. In some of these studies[16], ambiguous inclusion is defined such that an itemset $P$ is included in a transaction $T$ if $|P \cap T|/|P| \geq \theta$. 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. On the other hand, the authors introduced an inclusion allowing a constant number of missing items, i.e., $|P \setminus T| \leq \theta$. This does not violate the monotonicity, thereby admits both apriori and backtrack with many related techniques developed for frequent itemset mining. However, it has a disadvantage that a transaction can miss only few items of large itemsets whereas almost all small itemsets will be frequent.

In some studies[5, 7–9], they considered that it is a fault if an item of the itemset is not included in a transaction, and treat mining pairs of an itemset and a transaction set such that there are few faults between their elements. Their algorithms find pairs with few faults, but they are not always minimum solutions.

In this paper, we address the problem from the algorithmic point of view, and model the problem in a different way. In the other words, the goal of this paper is to investigate the most simple and useful model of ambiguous frequency which allows fast computation. In the existing practice-based approach, the designed model often allows no fast algorithm. Heuristic approaches lose the completeness and exactness of the algorithm. For developing fast algorithms, we consider another model for ambiguous frequency.

For an itemset $P$ and a transaction $T$, the *inclusion ratio* is the ratio of items of $P$ included in $T$, i.e., $|T \cap P|/|P|$. For an itemset $P$ and a transaction set $\mathcal{T}$, the *average inclusion ratio* of $\mathcal{T}$ for $P$ is defined by the average of the inclusion ratio of transactions in $\mathcal{T}$, i.e., $(\sum_{T \in \mathcal{T}} |T \cap P|)/(|\mathcal{T}||P|)$. By representing the inclusion between transactions and items by matrix, the average inclusion ratio corresponds to the density of the submatrix induced by the items and

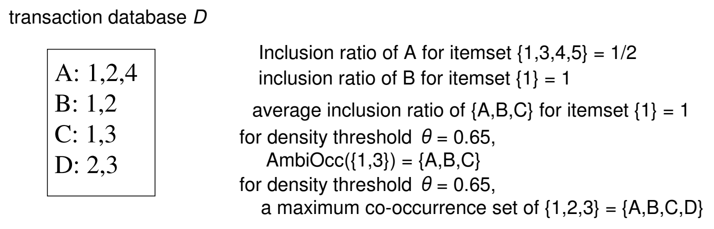
transaction database $D$
A: 1,2,4
B: 1,2
C: 1,3
D: 2,3
Inclusion ratio of A for itemset {1,3,4,5} = 1/2
inclusion ratio of B for itemset {1} = 1
average inclusion ratio of {A,B,C} for itemset {1} = 1
for density threshold $\theta$ = 0.65,
AmbiOcc({1,3}) = {A,B,C}
for density threshold $\theta$ = 0.65,
a maximum co-occurrence set of {1,2,3} = {A,B,C,D}

**Fig. 1.** Examples of average inclusion ratio and maximum occurrence sets

transactions. When the average inclusion ratio is high, the items co-occur in the transactions, or the transactions co-occur in the items. For a density threshold $\theta$, a transaction set of the largest size having average inclusion ratio no less than $\theta$ is called the *maximum co-occurrence set* for $P$. Note that any maximum co-occurrence set can be obtained by choosing transactions in decreasing order of their inclusion ratio. The size of the maximum co-occurrence set is called the *maximum co-occurrence size* of $P$, and is denoted by $cov(P)$. We denote the lexicographical minimum maximum co-occurrence set by $AmbiOcc(P)$. Some examples are shown in Figure 1.

For the minimum support threshold $\sigma$, an itemset is called an *ambiguous frequent itemset* if its maximum co-occurrence size is no less than $\sigma$. The problem in this paper is formulated as follows.

**Ambiguous Frequent Itemset Enumeration Problem**
**Input**: transaction database $\mathcal{D}$, minimum support $\sigma$, density threshold $\theta$
**Output**: all ambiguous frequent itemsets in $\mathcal{D}$

We propose a polynomial delay polynomial space algorithm, and show the practical performance by computational experiments. Note that an algorithm is *polynomial delay* if the computation time between any two consecutive solutions is polynomial in the input size.

If we represent the inclusion relation by a bipartite graph, an ambiguous frequent itemset and its corresponding transaction set corresponds to a vertex set inducing a dense bipartite subgraph, which is a quasi bipartite clique. Enumerating dense subgraphs whose edge density is no less than a threshold value can be done in polynomial delay polynomial space[14]. However, since an ambiguous frequent itemset has a lower bound for transaction sets, and identifies the same itemsets with different transaction sets, a direct application of the algorithm to our problem is not polynomial delay.

The existence of a polynomial delay algorithm for the enumeration problem of ambiguous frequent itemset is not trivial, since as we will show, simple algorithms involve an NP-complete problem in each iteration. The framework of the algorithm in this paper is motivated from the enumeration algorithm for pseudo cliques[14]. We introduce an adjacency relation with respect to a removal of an item between ambiguous frequent itemsets, and implicitly construct a tree-shaped traversal route on the relation. Our algorithm searches traverses the tree in a depth-first search manner, so that the computation time is polynomial delay. To best of our knowledge, this is the first result of even an output polynomial time algorithm for this problem. The ambiguous frequency and our algorithm can be naturally extended to a weighted version, in a straightforward manner.

## 2 Polynomial Delay 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. The repeated addition admits any ordering of items, hence we can efficiently avoid the duplications by adding items only in increasing order of their indices. Thus, we can construct a backtrack algorithm of a polynomial delay polynomial space. Precisely, the computation time for each frequent itemset is linear in the size of the database, i.e., $O(||\mathcal{D}||)$, where $||\mathcal{D}||$ is the size of database $\mathcal{D}$, i.e., $||\mathcal{D}|| = |\mathcal{D}| + \sum_{T\in\mathcal{D}} |T|$. The space complexity is optimal, that is, $O(||\mathcal{D}||)$.

However, the family of ambiguous frequent itemsets does not have this monotone property. For the database $\mathcal{D}$ in Figure 1, $\theta = 65\%$ and $\sigma = 4$, we can see that $cov(\{1,2,3\}) = 4$, as obtained by transaction set $\{A, B, C, D\}$, thereby $\{1,2,3\}$ is an ambiguous frequent itemset. However, the maximum co-occurrence size of its subset $\{1,3\}$ is 3, obtained by transaction set $\{A, B, C\}$, thereby $\{1,3\}$ is not an ambiguous frequent itemset. Since the monotonicity is not held, a straightforward backtrack algorithm is not applicable to the enumeration.

We approach the problem in another way. For itemset $P \neq \emptyset$, we define $e^*(P)$ by the item $e \in P$ that minimizes $|AmbiOcc(P) \cap Occ(\{e\})|$. Ties are broken by choosing the minimum index one. Using $e^*$, we introduce an adjacency relation among ambiguous frequent itemsets, and construct an implicit traversal route.

**Lemma 1.** *For any itemset $P \neq \emptyset$, there exists an item $e \in P$ satisfying $cov(P \setminus \{e\}) \geq cov(P)$.*

*Proof.* First we observe that the average inclusion ratio of $AmbiOcc(P)$ for $P$ is given by the average of $|AmbiOcc(P) \cap Occ(\{e\})|/|AmbiOcc(P)|$, since the average inclusion ratio of $AmbiOcc(P)$ for $P$ is $\frac{\sum_{e\in P} |AmbiOcc(P)\cap Occ(\{e\})|}{(|P|-1)\times|AmbiOcc(P)|}$. From the observation, the average inclusion ratio of $AmbiOcc(P)$ for $P \setminus \{e^*(P)\}$ is the average of $|AmbiOcc(P) \cap Occ(\{e\})|/|AmbiOcc(P)|$ among $P \setminus \{e^*(P)\}$, thus it is no less than the average inclusion ratio of $AmbiOcc(P)$ for $P$. It means that $cov(P \setminus \{e^*(P)\}$ is no less than $cov(P)$, thus $e^*(P)$ satisfies the condition to be the item $e$ in the statement. $\square$

For an itemset $P \neq \emptyset$, we define the parent $Prt(P)$ of $P$ by $P \setminus \{e^*(P)\}$. From Lemma 1, $P \setminus \{e^*(P)\}$ is an ambiguous frequent itemset. Particularly, $cov(Prt(P)) \leq cov(P)$. The cardinality of $Prt(P)$ is exactly one smaller than $P$, thus the parent child relation induced by $Prt$ is acyclic. Since every ambiguous frequent itemset other than the emptyset has a parent, the relation induces

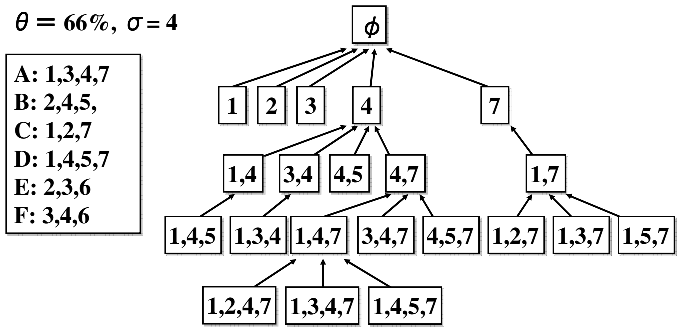

**Fig. 2.** Example of an enumeration tree

a rooted tree spanning all ambiguous frequent itemsets. We call this tree the *enumeration tree* of ambiguous frequent itemsets. An example of the enumeration tree is shown in Figure 2. By traversing the tree, we can find all ambiguous frequent itemsets without duplication.

To perform a depth-first search on the enumeration tree, we need to find all children of the current visiting ambiguous frequent itemset. By recursively finding the children, we can perform a depth-first search without using additional memory on visited vertices. This ensures the polynomiality of the memory complexity. The algorithm can be written as follows.

```
ReverseSearch($P$)
1. Output $P$
2. for each $e \notin P$
 2-1. if $P \cup \{e\}$ is an ambiguous frequent itemset then
 2-2.    if $Prt(P \cup \{e\}) = P$ then
 2-3.       call ReverseSearch $(P \cup \{e\})$
```

The computation of the average inclusion ratio and the parent of $P \cup \{e\}$ in 2-1 and 2-2 can be done in $O(||\mathcal{D}||)$ time. They are executed at most $n$ times in an iteration, thus the computation in an iteration except for those in the recursive calls is bounded by $O(||\mathcal{D}|| \times n)$. This algorithm outputs an ambiguous frequent itemset in each iteration, thus the computation time per ambiguous frequent itemset is $O(||\mathcal{D}|| \times n)$. The depth of the enumeration tree is bounded by $n$, thus we obtain the following theorem.

**Theorem 1.** *For given a transaction database $\mathcal{D}$, minimum support threshold $\sigma$, and density threshold $\theta$, ambiguous frequent itemsets in $\mathcal{D}$ can be enumerated in polynomial delay with polynomial space in terms of $||\mathcal{D}||$. In particular, the computation time for one ambiguous frequent itemset is $O(n||\mathcal{D}||)$ where $n$ is the number of items included in some transactions in $\mathcal{D}$.*

## 3 Improvements for Efficient Practical Computation

For practical huge databases, the computation time $O(||\mathcal{D}|| \times n)$ is quite long. It is not easy to reduce the time complexity, but possible to improve the practical efficiency by using at typical structures of actual datasets. The heavy tasks in each iteration with respect to itemset $P$ is the computation of $cov(P \cup \{e\})$ and $e^*(P \cup \{e\})$ for each $e$. Both need $O(n|\mathcal{D}|)$ time, and we will describe techniques to reduce the computation time for each. Note that the computation of $cov(P)$ is maybe heavier since we have to compute $e^*(P \cup \{e\})$ only when $P \cup \{e\}$ is an ambiguous frequent itemset.

We define $Occ_{=h}(P)$ by the set of transactions not including exactly $h$ items in $P$, i.e., $Occ_{=h}(P) = \{T \mid T \in \mathcal{D}, |P \setminus T| = h\}$. Similarly, $Occ_{\le h}(P) = \{T \mid T \in \mathcal{D}, |P \setminus T| \le h\}$. For the computation of $cov(P \cup \{e\})$, we have to obtain $Occ_{=h}(P \cup \{e\})$ for each $e$ and $h$, in increasing order of $h$. We can use the following property and lemma for efficient computation.

*Property 1.* [13] For a transaction $T$ included in $Occ_{=h}(P)$ for some $h, 0 \le h \le k$, $T \in Occ_{=h}(P \cup \{i\})$ holds if $T$ includes $i$. Otherwise, $T \in Occ_{=h+1}(P \cup \{i\})$.

**Lemma 2.** *[13] (a) $Occ_{=0}(P \cup \{i\}) = Occ_{=0}(P) \cap Occ(\{i\})$, and (b) $Occ_{=h}(P \cup \{i\}) = (Occ_{=h}(P) \cap Occ(\{i\})) \cup (Occ_{=h-1}(P) \setminus Occ(\{i\}))$ for any $h \ge 1$.*

From these, we can see that $Occ_{=h}(P \cup \{e\})$ for all $h$ are obtained by moving transactions of $Occ_{=h}(P) \cap Occ(\{e\})$ to $Occ_{=h+1}(P)$. This takes $O(|Occ(\{e\})|)$ time, which is expected to be small when the input database is sparse. To compute $Occ_{=h}(P) \cap Occ(\{e\})$, a method called *delivery* is efficient[11–13].

We briefly explain the framework of Delivery. An example is shown in Fig. 3. First, we prepare an empty bucket for each item $e$. Next, for each transaction $T$ in $Occ_{=h}(P)$, we "insert $T$ into the bucket of $e$ for each item $e \in T$". After performing this operation for all transactions in $Occ_{=h}(P)$, the content of the bucket of $e$ is equal to $Occ_{=h}(P \cup \{e\})$. The pseudo code of occurrence deliver is described as follows. The code inputs a transaction set $\mathcal{S}$, then sets $bucket[e]$ to $\mathcal{S} \cap Occ(\{e\})$ for all $e$. We suppose that the bucket of any item $e$ is initialized, and thus is empty at the beginning.

```
Delivery($\mathcal{S}$)
1. for each $T \in \mathcal{S}$ do
2.    for each $i \in T$, insert $T$ into $bucket[i]$
```

**Lemma 3.** *[12, 11]* **Delivery** *computes $\mathcal{S} \cap Occ(\{e\})$ for all $e$ in $O(||\mathcal{S}||)$ time.*

Let $k^*(P)$ be the smallest $h$ satisfying $AmbiOcc(P) \subseteq Occ_{\le h}(P)$.

**Lemma 4.** *If $P \cup \{e\}$ is a child of $P$, $k^*(P \cup \{e\}) \le k^*(P) + 1$ holds and $Occ_{\le k^*(P)}(P)$ includes a maximum co-occurrence set of $P \cup \{e\}$.*

*Proof.* From Lemma 2, we have $Occ_{\le k^*(P)}(P \cup \{e\}) \subseteq Occ_{\le k^*(P)}(P) \subseteq Occ_{\le k^*(P)+1}(P \cup \{e\})$. This means that $Occ_{\le k^*(P)}(P)$ is constructed by taking $|Occ_{\le k^*(P)}(P)|$ transactions from $Occ_{\le k^*(P)+1}(P \cup \{e\})$ in decreasing order of the inclusion ratio for $P \cup \{e\}$. If $P \cup \{e\}$ is a child of $P$, we have $cov(P \cup \{e\}) \le cov(P) \le |Occ_{\le k^*(P)}(P)|$. Thus, $Occ_{\le k^*(P)}(P)$ includes a maximum co-occurrence set of $P \cup \{e\}$, and $k^*(P \cup \{e\}) \le k^*(P) + 1$. □

This lemma implies that for the computation of $cov(P\cup\{e\})$, we have to look only at the transactions included in $Occ_{\le k^*(P)+1}(P)$. This reduces the computation time to $O(||Occ_{\le k^*(P)+1}(P)||)$, where $||Occ_{\le k^*(P)+1}(P)||$ is the sum of the sizes of transactions in $Occ_{\le k^*(P)+1}(P)$.

We next state a lemma to determine $k^*(P \cup \{e\})$ efficiently. Let $Th(P, k) = \theta \times (|P| + 1) \times |Occ_{\le k}(P)| - \sum_{T \in Occ_{\le k}(P)} |T \cap P|$. If and only if $|Occ_{\le k}(P) \cap Occ(\{e\})| \ge Th(P, k)$, the average inclusion ratio of $Occ_{\le k}(P)$ for $P \cup \{e\}$ is no less than $\theta$. For any transactions $T \in Occ_{=h}(P)$ and $T' \in Occ_{=h+1}(P)$, the inclusion ratio of $T$ for $P \cup \{e\}$ is always no less than that of $T'$ for $P \cup \{e\}$. Thus, we have the following property.

**Lemma 5.** *Suppose that $P\cup\{e\}$ is a child of $P$. Then, both $|Occ(\{e\})\cap Occ_{\le k-1}(P)| \ge Th(P, k-1)$ and $|Occ(\{e\}) \cap Occ_{\le k}(P)| < Th(P, k)$ hold if and only if $|Occ_{\le k-1}(P)| < cov(P \cup \{e\}) \le |Occ_{\le k}(P)|$.*

Note that the statement holds for a unique $k$ since $Th(P, k)$ is monotone decreasing for the increase of $k$. From the above lemma, we compute $|Occ(\{e\}) \cap Occ_{\le k-1}(P)|$ in increasing order of $k$ from $k = 1$, then find each item $e$ satisfying the condition of Property 5 with $k$, and check whether $P \cup \{e\}$ is a child of $P$ or not by computing $AmbiOcc(P \cup \{e\})$. The algorithm based on this method is as follows.

**ALGORITHM** FindAllChildren($P$)
```
1. compute $Th(P,k)$ for each $k = 0, ..., k^*(P) + 1$
2. for $k = 0$ to $k^*(P)$
3.   compute $Occ_{=k}(P) \cap Occ(\{e\})$ for each $e$
4.   for each $e$ s.t. $|Occ(\{e\}) \cap Occ_{\le k-1}(P)| \ge Th(P, k-1)$ do (for each $e$ if $k = 0$)
5.       if $|Occ(\{e\}) \cap Occ_{\le k}(P)| < Th(P, k)$ then
6.         if $\sigma \le |AmbiOcc(P \cup \{e\})| \le |AmbiOcc(P)|$ then
7.           if $Prt(P \cup \{e\}) = P$ then $P \cup \{e\}$ is a child
8.   end for
9. end for
```

Step 6 computes $AmbiOcc(P \cup \{e\})$, then obtain $Prt(P \cup \{e\})$ by computing $AmbiOcc(P\cup\{e\})\cap Occ\{e'\}$ for each $e' \in P\cup\{e\}$. At the time of computing these values, we already have computed $Occ_{\le k^*(P\cup\{e\})-1}(P)\cap Occ_{\le k^*(P\cup\{e\})}(P\cup\{e\})$, thus we have to compute only $Occ_{=k^*(P\cup\{e\})}(P) \cap Occ(\{e\})$.

Now the computation time in each iteration with respect to $P$ is (a) $O(||Occ_{\le k^*(P)+1}(P)||)$ for computing $cov(P \cup \{e\})$ for all $e$, and (b) $O(||Occ_{=k^*(P\cup\{e\})}(P)||)$ for each $e$ such that $P \cup \{e\}$ is an ambiguous frequent itemset. In practical datasets, it is expected that $P\cup\{e\}$ is an ambiguous frequent

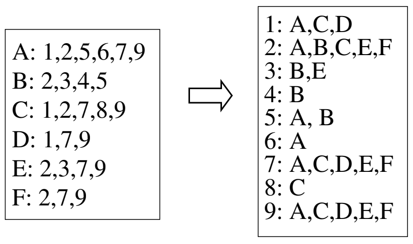

**Fig. 3.** Example of delivery

itemset only for few $e$'s. Otherwise the number of ambiguous frequent itemsets is huge so that we can not enumerate them in a practically short time, and we can not deal with huge output itemsets. Therefore, we can expect that (b) is not larger so much than (a), thus the computation time for an iteration is $O(||Occ_{\leq k^*(P)+1}(P)||)$, which is relatively shorter than $O(n||\mathcal{D}||)$.

## 4 Weighted Ambiguous Frequent Itemset

In practical transaction databases, items of each transaction often has several different weights. For example, POS data includes the number or the price of each item purchased by a customer. In experiments in industry or natural science, each cell or item may have a kind of intensity. Such a database can be regarded as a matrix of item columns and transaction rows such that each cell has a value. One may be naturally motivated to find submatrices with a large average weight of cells. These locally heavy submatrices correspond to important objects such as clusters, and have applications in knowledge discovery and data engineering.

We define the problem as follows. We suppose that each item $e$ of a transaction $T$ has a weight $w(T,e)$. For an itemset $P$ and a transaction $T$, we define the *average weight* $w(T,P)$ of $T$ with respect to $P$ by $(\sum_{e\in P\cap T} w(T,e))/|P|$. For a set $\mathcal{T}$ of weighted transactions, we define the *average weight* $w(\mathcal{T},P)$ by $(\sum_{T\in\mathcal{T}} w(T,P))/|\mathcal{T}|$. When we are given a weight threshold $\theta$, we define the *weighted maximum co-occurrence size* of $P$ by the maximum size of a transaction set having average weight no less than $\theta$. For a given support threshold $\sigma$, an itemset is called a *weighted ambiguous frequent itemset* if its weighted maximum co-occurrence size is no less than $\sigma$. The weighted version of the ambiguous frequent itemset enumeration problem is to output all weighted ambiguous frequent itemsets. Given these definitions, we obtain a similar neighboring relation between weighted ambiguous frequent itemsets.

**Theorem 2.** *For given a weighted transaction database $\mathcal{D}$, weight threshold $\theta$, and minimum support threshold $\sigma$, all weighted ambiguous frequent itemsets can be enumerated in polynomial delay and polynomial space. In particular, the computation time is $O(||\mathcal{D}||n)$ for each, and the space complexity is linear in the input size, where $n$ is the number of items in $\mathcal{D}$.*

The method described in the above sections is not directly applicable to improve the practical efficiency of the weighted version of our algorithm. The reason is that Properties 1 and 2 are not valid for the weighted version. To compute the weighted maximum co-occurrence size of $P \cup \{e\}$, we need to get the transactions $T$ in the order of $w(T, P \cup \{e\})$. If $w(T, P \cup \{e\})$ is large, then either $w(T, P)$ or $w(T, \{e\})$ has to have a large value. Thus, by getting transactions having large average weights with respect to either $P$ or $\{e\}$, we can efficiently compute the weighted maximum co-occurrence size.

## 5 Hardness Result for Branch-and-Bound Approaches

We show that a hardness result for simple approaches to answer the question that why we need a sophisticated enumeration scheme. In a typical branch-and-bound algorithm, we may choose an item $e$ and divide the enumeration problem into two subproblems; the enumeration problem of ambiguous frequent itemsets including $e$, and the problem for itemsets not including $e$. The division of the problem is done recursively until the problem includes a unique solution (ambiguous frequent itemset). In this approach we have to know the existence of solutions to the restricted problem, otherwise we will divide problems having no solution recursively, thereby exponentially many times.

The following theorem states that this problem is NP-complete. Therefore, we observe that it is hard to get a polynomial delay algorithm by typical branch-and-bound since we have to solve an NP-complete problem in each iteration.

**Theorem 3.** *For given a transaction database $\mathcal{D}$, itemset $S$, density threshold $\theta$, and minimum support threshold $\sigma$, the problem of answering whether an ambiguous frequent itemset including $S$ exists or not, is NP-complete.*

*Proof.* Suppose that we are given a transaction database $\mathcal{D}$, a minimum support threshold $\sigma$, and a constant number $k$, and going to check for the existence of an itemset of size at least $k$ that is included in at least $\sigma$ transactions. This is known to be NP-complete[17]. Let $I$ be the set of items included in transactions in $\mathcal{D}$, and $I'$ be a set of items of size $|\mathcal{D}| \times |I|$ satisfying $I \cap I' = \emptyset$. We choose an item $e^*$ from $I'$.

We now construct a transaction database $\mathcal{D}' = \{T \cup (I' \setminus \{e^*\}) | T \in D\}$. Let $X$ be a subset of $I$, $\mathcal{T}$ be a transaction set of $D$, and $\mathcal{T}'$ be the transaction set of $\mathcal{D}'$ corresponding to $\mathcal{T}$. Then, $X$ is a frequent itemset of $\mathcal{D}$ and $\mathcal{T} = Occ(X)$ if and only if the average inclusion ratio of $\mathcal{T}'$ for $X \cup I'$ is strictly larger than $(|\mathcal{D}| \times |I| - 1)/(|\mathcal{D}| \times |I|)$. In particular, when $|X| = k$, the average inclusion ratio is $(|\mathcal{D}| \times |I| + k - 1)/(|\mathcal{D}| \times |I| + k)$. Here we set $\theta = (|\mathcal{D}| \times |I| + k - 1)/(|\mathcal{D}| \times |I| + k)$. Then, $X$ is a frequent itemset of $\mathcal{D}$ of size at least $k$ if and only if $X \cup I'$ is an ambiguous frequent itemset of $\mathcal{D}'$. Therefore we have the theorem. □

## 6 Computational Experiments

In general, the practical computation time of an algorithm often differs from the theoretical upper bound. The reason is that the computation time is dominated

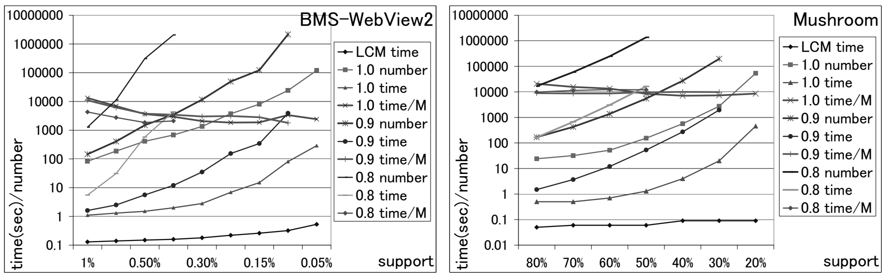

**Fig. 4.** Computation time and #solutions on BMS-WebView and Mushroom

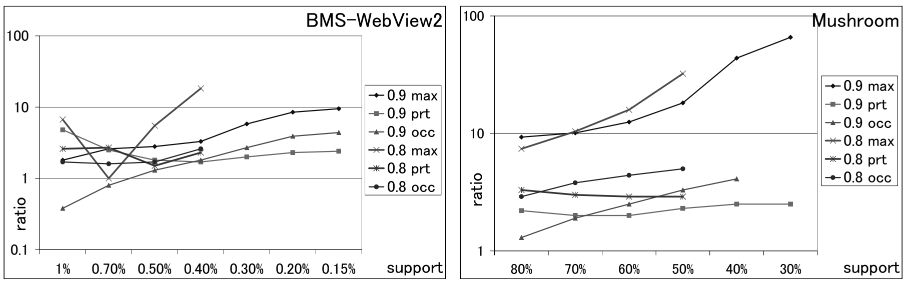

**Fig. 5.** Comparison of accessed items on BMS-WebView and Mushroom

by the "average", but the theoretical upper bound looks only at the worst case. To see the gap and to be a help for the practical use, we show the results of some computational experiments.

The C code is used for the implementation. The computer used in the experiments was a notebook PC with a Pentium M 1.1GHz processor with 768MB memory. The experiments were done on cygwin which is an emulator of Linux environments on Windows. The implementation is a simpler version of our algorithm, which compute the parent in a straightforward way. The reason is to choose a simpler version is to see the performance of a simple implementation, which would help for coding. The implementation is available at the author's homepage; http://research.nii.ac.jp/˜uno/index.html.

We examined two practical datasets taken from FIMI repository[6]. The first is BMS-WebView2 with about 3,300 items and 77,000 transactions. The average size of transactions is 4.6, thus the dataset is quite sparse. The second is Mushroom with about 120 items and 8,000 transactions. The average size of transactions is 23, thus the dataset is not sparse.

We run the implementations with the thresholds $\theta = 0.8, 0.9$ and 1.0. Since we could not find any implementation for the ambiguous frequent itemset enumeration, we have no comparison to other implementations. Instead of this, we compare the performance to that of an ordinary frequent itemset enumeration algorithm LCM[12, 11]. Since the frequent itemset enumeration is a special case of our problem, it can be considered as a kind of upper bound of the performance of the ambiguous frequent itemset enumeration.

The results are shown in Fig. 4. The left is BMS-WebView2, and the right is Mushroom. The horizontal axis is for minimum support threshold, and the vertical axis is for computation time, computation time for 1 million (ambiguous) frequent itemsets, and the number of output itemsets, written in log scales.

The computation time of our algorithm increases as the decrease of minimum support, but the computation time per one million itemsets does not change drastically. It seems to change as the change of average size of $Occ(\{e\})$. Comparing to the ordinary frequent itemset mining algorithm, the performance of our algorithm is not so good. One of the reason is that the cost for computing parents of the candidate children. A simple duplication check by storing the discovered itemsets in memory will accelerate the computation when the output itemsets are few. The other reason is that in the ordinary frequent itemset mining, we can use the conditional database for the current operating itemset, which includes only items larger than the maximum item in the current operating itemset and are frequent in the database induced by the occurrence of the current operating itemset. Usually the number of items in the conditional database is much smaller than the number of items in the original database, thus the computation is faster. To reduce the difference on the computation time, further techniques for the efficient computation are still needed. The number of ambiguous frequent itemsets increases drastically by the decrease of density threshold. In practice, we should use a threshold slightly smaller than 1.0.

We also looked at several statistics on the experiments in Figure 5. "max" means the ratio of ambiguous frequent itemsets and the number of maximal ambiguous frequent itemsets to which no item addition yields an ambiguous frequent itemset. "prt" shows the ratio of the number of accessed items between a straightforward algorithm and the sophisticated algorithm proposed in this paper, and "occ" indicates that between delivery for computing the frequencies of all additions of items, and delivery for computing the parent. As we can see, these ratio increase as the increase the number of solutions. Thus, we can expect the decrease of the number of solutions by outputting only maximal ones. The speedup is also expected by introducing our sophisticated parent computation, but the effect will be limited. The big ratio of "occ" implies that the big gap between computation time of our algorithm and ordinary frequent itemset mining. It also implies that more practical improvements are needed.

## 7 Conclusion and Future Work

We formulated the enumeration problem of ambiguous frequent itemsets, and proposed a polynomial delay polynomial space algorithm. The algorithm is naturally extended to a weighted version. The experimental performance for practical datasets is acceptable, but improvements on practical performance is a crucial future work. Another interesting research topic is extending the technique to other frequent pattern mining problems.

## Acknowledgments

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 a joint-research fund of 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, *Disc. App. Math.* 65, pp. 21–46, 1996.
4. R. J. Bayardo Jr., Efficiently Mining Long Patterns from Databases, *SIGMOD98*, 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, LNCS* 3377, pp. 33–45, 2005.
6. B. Goethals, the FIMI repository, `http://fimi.cs.helsinki.fi/`, 2003.
7. J. Liu, S. Paulsen, W. Wang, A. Nobel, J. Prins, Mining Approximate Frequent Itemsets from Noisy Data," *ICDM05*, pp. 721-724, 2005
8. J. K. Seppanen and H. Mannila, Dense Itemsets, *SIGKDD 2004*, 2004.
9. W. Shen-Shung and L. Suh-Yin, Mining Fault-Tolerant Frequent Patterns in Large Databases, *ICS2002*, 2002.
10. M. Takeda, S. Inenaga, H. Bannai, A. Shinohara, and S. Arikawa, Discovering Most Classificatory Patterns for Very Expressive Pattern Classes, *LNCS* 2843, pp. 486–493, 2003.
11. T. Uno, T. Asai, Y. Uchida, H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, *LNAI* 3245, pp. 16–31, 2004.
12. T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets, *IEEE ICDM'04 Workshop FIMI'04*, 2004.
13. T. Uno, H. Arimura, An Efficient Polynomial Delay Algorithm for Pseudo Frequent Itemset Mining, *LNAI* 4755, pp. 219–230, 2007
14. T. Uno, An Efficient Algorithm for Enumerating Pseudo Cliques, *ISAAC 2007, LNCS* 4835, pp. 402–414, 2007.
15. 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, *SIGMOD 94*, pp. 115–125, 1994
16. C. Yang, U. Fayyad, P. S. Bradley, Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions, *SIGKDD* 2001, 2001.
17. M. J. Zaki and M. Ogihara, Theoretical foundations of association rules, *In 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998.