TCS-TR-A-14-72

TCS Technical Report

Fast Statistical Assessment for Combinatorial Hypotheses Based on Frequent Itemset Mining

by

Shin-ichi Minato, Takeaki Uno, Koji Tsuda,
Aika Terada, and Jun Sese

Division of Computer Science

Report Series A

April 22, 2014

Hokkaido University
Graduate School of
Information Science and Technology

Email: minato@ist.hokudai.ac.jp

Phone: +81-011-706-7682

Fax: +81-011-706-7682

Fast Statistical Assessment for Combinatorial Hypotheses Based on Frequent Itemset Mining

Shin-ichi Minato[*]
Grad. School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060–0814, Japan

Takeaki Uno
National Institute of Informatics
Tokyo 101–8430, Japan

Koji Tsuda[†]
Grad. School of Frontier Sciences,
The University of Tokyo
Kashiwa, 277–8561 Japan

Aika Terada
Grad. School of Humanities and Sciences,
Ochanomizu University
Tokyo, 112–8610 Japan,

Jun Sese
Grad. School of Humanities and Sciences,
Ochanomizu University
Tokyo, 112–8610 Japan,

April 22, 2014

(Abstract) In many scientific communities using experiment databases, one of the crucial problem is how to assess the statistical significance (P-value) of a discovered hypothesis. Especially, combinatorial hypothesis assessment is a hard problem because it requires a multiple-testing procedure with a very large factor of the P-value correction. Recently, Terada et al. proposed a novel method of the P-value correction, called "Limitless Arity Multiple-testing Procedure" (LAMP), which is based on frequent itemset enumeration to exclude meaninglessly infrequent itemsets which never be significant. The LAMP makes much more accurate P-value correction than previous method, and it empowers the scientific discovery. However, the original LAMP implementation is sometimes too time-consuming for practical databases. We propose a new LAMP algorithm that essentially executes itemset mining algorithm once, while the previous one executes many times. Our experimental results show that the proposed method is much (10 to 100 times) faster than the original LAMP. This algorithm enables us to discover significant P-value patterns in quite short time even for very large-scale databases.

[*]He also works for JST ERATO Minato Project.

[†]He also works for JST ERATO Minato Project and National Institute of Advanced Industrial Science and Technology.

1 Introduction

Discovering useful knowledge from large-scale databases has attracted considerable attention during the last decade. Such knowledge discovery techniques are widely utilized in many areas of experimental sciences, such as biochemistry, material science, medical science, etc. In those scientific communities using experiment databases, one of the crucial problem is how to assess the statistical significance (P-value) of a discovered hypothesis. The P-value-based assessment is one of the most important factors in the paper review process of academic journals in experimental sciences [9].

For those scientific applications, detecting a combinatorial regulation of multiple factors is sometimes a very important issue. For example, it is well known that a key to generate iPS cells consists of the four factors of genes [13]. However, statistical assessment of a hypothesis for detected combinatorial effect is a hard problem because it requires a multiple-testing procedure with a very large factor of the P-value correction. This correction is necessary to avoid a false discovery caused by repetition of statistical tests. When we consider the combinations of j out of n hypotheses, the number of tested combinations increases exponentially as $O(n^j)$, and if we use a naive correction method in $j = 4$ or more, it is too conservative since almost all discoveries becomes extremely unlikely.

Recently, Terada et al. developed a novel P-value correction procedure, called "Limitless Arity Multiple-testing Procedure" (LAMP). Their paper [16] was published in *PNAS*, a leading journal in scientific community. This new procedure excludes meaninglessly infrequent hypotheses which never be significant. The P-value correction factor calculated by LAMP is much more accurate than previous method, and it empowers the scientific discovery from the experiment databases. However, the original LAMP implementation is sometimes too time-consuming for practical databases, and a state-of-the-art algorithm has been desired.

The LAMP is based on the techniques of *frequent itemset mining*, to enumerate all frequent itemsets included in at least σ transactions of the database for a given threshold σ. Since the pioneering work by Agrawal *et al.* [1], various algorithms have been proposed to solve this problem [7, 11, 22]. Among those state-of-the-art algorithms, *LCM (Linear time Closed itemset Miner)*[20, 18, 19] by Uno et al. is known as one of the fastest algorithm, which is based on a depth-first traversal of a search tree for the combinatorial space.

In this paper, we propose a fast itemset enumeration algorithm to find the minimum support for satisfying the LAMP condition. Our new algorithm essentially executes itemset mining algorithm once, while the previous one executes many times. We show that LAMP condition is a kind of threshold function which is monotonically decreasing or increasing. We developed a general scheme to explore the maximum frequency satisfying a given threshold function. We successfully applied this new scheme to the LAMP condition. Those new procedures are implemented into the newest version of the LCM program. Our experimental results show that the proposed method is much (10 to 100 times) faster than the original LAMP. This algorithm enables us to discover significant P-value patterns in quite short time even for very large-scale databases.

In the rest of this paper, we first explain the preliminaries on frequent itemset mining algorithms. In Section 3, we then present the problem of statistical assessment for combinatorial hypotheses and the idea of LAMP. Section 4 describes our proposed methods for finding the minimum frequency for satisfying threshold function and the LAMP condition. Section 5 shows our experimental results, followed by conclusion.

2 Preliminary

Here we start with some basic definitions of itemset databases and frequent itemset mining.

Let $E = \{1, \ldots, n\}$ be a set of items. A subset of E is called an *itemset*. A *transaction database* is a database composed of *transactions* where a transaction is an itemset. A transaction database can include two or more identical transactions. For a transaction database $\mathcal{D}$, $|\mathcal{D}|$ denotes the number of transactions in $\mathcal{D}$, and $||\mathcal{D}||$ denotes the size of $\mathcal{D}$, that is the sum of the size of the transactions in $\mathcal{D}$, i.e., $||\mathcal{D}|| = \sum_{T \in \mathcal{D}} |T|$.

For an itemset X and a transaction database $\mathcal{D}$, an *occurrence* of X in $\mathcal{D}$ is a transaction including X. The *occurrence set* of X, denoted by $Occ(X)$ is the set of all occurrences of X in $\mathcal{D}$. The *frequency* of X is the number of occurrences of X, and is denoted by $frq(X)$. For a given constant number σ called *minimum support*, an itemset X is called *frequent*. The frequent itemset mining problem is to enumerate all frequent itemsets for given database $\mathcal{D}$ and threshold σ.

Without confusions, an item e also represents the itemset $\{e\}$, hence $frq(e)$, $X \cup e$ and $X \setminus e$ denote $frq(\{e\})$, $X \cup \{e\}$ and $X \setminus \{e\}$, respectively. Let $\kappa(\sigma)$ be the number of frequent patterns whose frequencies are no less than σ.

2.1 Frequent Itemset Mining Algorithm

The set system given by the set of frequent itemsets is anti-monotone, i.e., any subset Y of a frequent itemset X is always frequent. Thus, enumeration of frequent itemsets is done efficiently by hill-climbing algorithms, that start from the emptyset and recursively add items unless the itemset is infrequent. In particular, depth-first search type algorithms (backtracking) are known to be efficient [7]. In the backtracking way, we add an item e to the current itemset X, and explore all itemsets generated from $X \cup e$ before processing $X \cup e', e' \neq e$. To avoid duplicated solutions such that an itemset is output twice, backtracking adds only item $e > tail(X)$ where $tail(X)$ is the maximum item in X. By this derive, any item Y is generated from the itemset $Y \setminus tail(X)$, thus no duplication occurs. The pseudo code of backtracking is written as follows.

$T \in \mathcal{D}$					$\mathcal{C}(T)$
	$\underline{2}$	$\underline{3}$	4	5	**neg**
1		3		5	pos
	$\underline{2}$	$\underline{3}$	4		**pos**
1	2		4	5	neg
1	$\underline{2}$	$\underline{3}$			**pos**
	2		4		neg
1	$\underline{2}$	$\underline{3}$		5	**pos**
	2		4	5	neg
1		3	4		neg

Figure 1: An example of transaction database with positive-negative class

	pos	neg	all
$Occ(X)$	(σ_p) 3	$(\sigma - \sigma_p)$ 1	(σ) 4
$\overline{Occ}(X)$	$(p - \sigma_p)$ 1	$(m - p - \sigma + \sigma_p)$ 4	$(m - \sigma)$ 5
all	(p) 4	$(m - p)$ 5	(m) 9

Figure 2: Contingency table ($X = \{2, 3\}$)

```
ALGORITHM **BackTracking_Basic** ($X$)
1. **output** $X$
2. **for** each item $e > tail(X)$,
   **if** $frq(X \cup e) \geq \sigma$ **then call BackTracking_Basic** ($X \cup e$)
```

The most heavy computation in this algorithm is the computation of $frq(X \cup e)$ on Step 2 of the algorithm. There are several techniques for reducing this computation. Especially, *recursive database reduction* techniques, such as FP-tree representation of the database [10] and anytime database reduction [18], are quite efficient. These techniques can be applied only to depth-first type algorithm, thus this is why breadth-first search type algorithm is slow compared to backtracking on real-world data [8]. There is one more important technique so called *equi-support* (see for example [20] written as *hypercube decomposition*). For an itemset X, we call an item $e \notin X$ *addible* if $frq(X \cup e) = frq(X)$. $Eq(X)$ denotes the set of addible items e of X that satisfy $e > tail(X)$. We can see that any itemset $X \cup S, S \subseteq Eq(X)$ satisfies $frq(X) = frq(X \cup S)$, thus we output all these itemsets without the computation of their frequency. Using this observation, a more efficient algorithm can be written as follows. Duplications can be avoided by not generating recursive calls for $X \cup e$ with $e \in Eq(X)$.

```
ALGORITHM **BackTracking** ($X$)
1. **output** $X \cup S$ for all $S \subseteq Eq(X)$
2. **for** each item $e > tail(X), e \notin Eq(S)$,
   **if** $frq(X \cup e) \geq \sigma$ **then call BackTracking** ($X \cup e$)
```

Fast implementations of backtracking find up to one million solutions in a second [8].

3 Statistical Assessment for Combinatorial Hypotheses

In this paper, we discuss a fast method of statistical assessment using the techniques of frequent itemset mining. First we assume the following experimental scenario.

Now we have a scientific database including experimental results for a number of human gene samples, and each sample shows a set of expressions of targeted factors. Here we assume only one expression level (exist or not) for each factor. We also have another classification result for each sample whether it is positive or negative, for example, the gene sample is given by a patient of breast cancer or a normal person. Then we want to discover a combination of factors which is highly correlated to incidence of a breast cancer. This is a quite simple scenario and many similar cases may commonly appear in various areas of experimental sciences. Since we assume only binary values in the database, we can represent it as a transaction database $\mathcal{D}$ as shown in Fig. 1. Here we also assume a classifier $\mathcal{C} : \mathcal{D} \rightarrow \{pos, neg\}$, which classify each transaction into a positive or a negative one. Suppose that the database has n items for the expressions of factors, m transactions in total, and p positive transactions.

Now we consider the assessment whether a given itemset has a strong correlation to appear in the positive transactions. Figure 2 shows a *contingency table* between the occurrence of the itemset X and the positive class of transactions. Here we note $\sigma = frq(X)$ and σ_p is a number of positive transactions in $Occ(X)$. We also show the value of each cell when $X = \{2, 3\}$ for the database shown in Fig. 1.

If the distribution of the contingency table is very biased, we may consider this is a kind of knowledge discovery because it is unlikely happened incidentally. The *P-value* represents the probability that such a biased distribution incidentally happens. In other words, it is the probability of a false discovery, and we can accept the statistical significance if the P-value is smaller than a threshold α. ($\alpha = 0.05$ is often used.) The P-value is quite important in the paper review process of academic journals in experimental sciences. *Fisher's exact test* is one of the major method for calculating the P-value for a given contingency table. This testing method assumes that each experimental result is independent and has an equal weight. Then, the P-value is calculated by counting all combinatorial cases to generate equally or more biased distributions of the contingency table.

3.1 P-value Correction in Multiple Tests

If we repeatedly calculate P-values for many different factors in a same database, we may more likely find a factor with an incidentally low P-value. For example, if we explore a family of hundred hypotheses each of which might be a false discovery in p=0.05, then at least one false discovery can be found in p=0.994. It is well known that such multiple tests may cause serious false positive problems [14].

Hence, multiple testing correction must be used in order to avoid a false discovery. The *family-wise error rate (FWER)* indicates the probability that at least one false discovery happens in multiple tests. This rate increases at most linearly as the number of tests increases, which motivates the Bonferroni correction [6] that multiplies the raw P-value by the number of tests. In other words, we must compare the raw P-values with the adjusted threshold $\delta = \alpha/k$, where k is number of tests. The Bonferroni correction is a very conservative method, which likely causes false negative but hardly causes false positive, and often used in academic articles in experimental sciences.

When we consider the combinations of j out of n items, the number of tested combinations increases exponentially as $O(n^j)$, and if we naively use Bonferroni correction in $j = 4$ or more, most of the discoveries of combinatorial hypotheses becomes extremely unlikely. However, in many practical cases, not all the combinations occur in the databases, so the ideal size of hypothesis family would be much smaller than the naive combinatorial number. This is a motivation of the LAMP.

3.2 Idea of LAMP

Recently, Terada et al. developed a novel P-value correction procedure, called "Limitless Arity Multiple-testing Procedure" (LAMP) [16]. This new procedure is based on frequent itemset enumeration for checking the two principles:

(1) Meaninglessly infrequent itemsets which never be significant can be excluded from the number of hypotheses to be counted.

(2) Any different itemsets having completely the same occurrence set can be counted as one hypothesis in the family.

The principle (2) means that we may enumerate only the *closed* itemsets, and we can just use existing state-of-the-art algorithms of closed itemset mining [20, 18, 19]. Here we explain in detail how to check the principle (1).

Suppose an itemset X with a very low frequency $\sigma(< p)$. Using Fisher's exact test, the raw P-value cannot be smaller than

$$f(\sigma) = \binom{p}{\sigma} / \binom{m}{\sigma}, \tag{1}$$

which means the most biased case that all $Occ(X)$ are classified into the positive class. Note that $f(\sigma)$ is monotonically decreasing, namely, the less frequent itemsets has the larger $f(\sigma)$. This observation means that all the infrequent itemsets satisfying $f(\sigma) > \delta$ can never be significant, regardless of the classification result. Thus, we can exclude such itemsets from the number of hypotheses to be counted. This is the key idea of the LAMP.

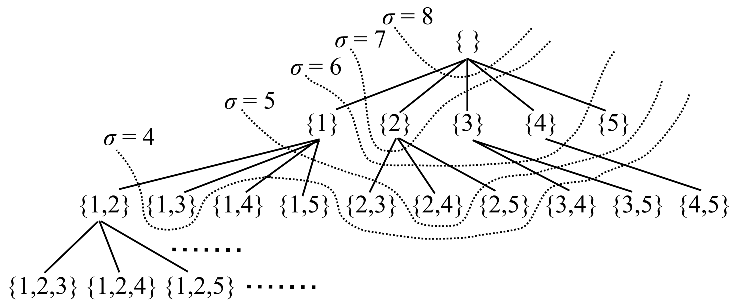

Figure 3: An example of a breadth-first search

Let $\kappa_c(\sigma)$ be the number of all closed itemsets not less frequent than σ. Then, the adjusted threshold of Bonferroni correction can be written as $\delta = \alpha/\kappa_c(\sigma)$. Now our subject is to find the maximum frequency σ_{max} which satisfies:

$$f(\sigma_{max} - 1) > \frac{\alpha}{\kappa_c(\sigma_{max})} \quad \text{and} \quad f(\sigma_{max}) \leq \frac{\alpha}{\kappa_c(\sigma_{max} + 1)} \quad . \tag{2}$$

3.3 Current Implementation of LAMP

In the above inequation of the LAMP condition, the left side $f(\sigma - 1)$ is monotonically decreasing and the right side $\alpha/\kappa_c(\sigma)$ is monotonically increasing in the range $1 \leq \sigma \leq p$. Thus, a naive idea is to enumerate all the itemsets in an order from the highest σ to the lowest one, and we may stop the enumeration procedure when the current σ first satisfies the condition. This is well known as the breadth-first search approach used in the "Apriori-type" [1] frequent itemset mining algorithms. Figure 3 illustrates the execution steps of a breadth-first search on the same database in Fig. 1. This approach requires a large memories to store all the itemsets at the current frontier for each step, and in general, it is not very efficient for practical size of databases.

The current implementation of the original LAMP [15] is not using a breadth-first search due to memory limitation, but just calls the LCM algorithm repeatedly, as follows.

```
ALGORITHM **Original_LAMP** ($\alpha$)
1. $\sigma := max_{e \in E}(frq(e))$
2. **if** $\sigma < p$ **then** $\sigma := p$     // $p$: number of positive transactions.
3. $k := \kappa_c(\sigma)$                 // call **LCM** algorithm to compute $\kappa_c(\sigma)$.
4. **if** $f(\sigma - 1) \leq \alpha/k$ **then** $\sigma := \sigma - 1$; **goto** 3
5. **output** $(\sigma, k)$
```

This method requires less memory, but sometimes very time-consuming since the LCM may be called more than thousand times for practical cases, and it may take a day or more. The current LAMP has a bottleneck in computation time, and some efficient algorithms are desired.

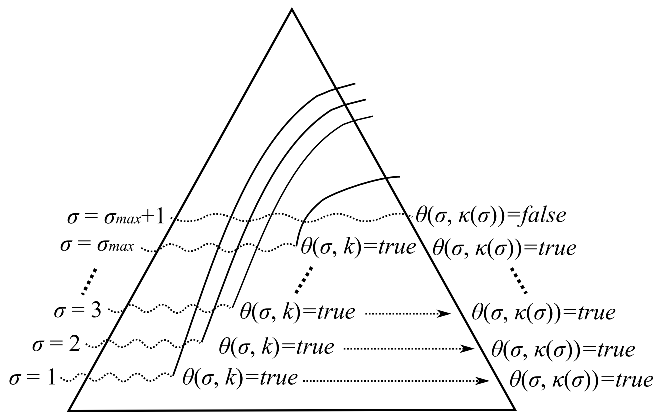

Figure 4: The scheme of support increase algorithm

4 Proposed Algorithms

4.1 A Threshold Function for the LAMP condition

Let $\theta(x, y) : (\mathbf{N} \times \mathbf{N}) \rightarrow \{true, false\}$ be a given threshold function such that $\theta(x, y) = true$ implies $\theta(x', y) = true$ for any $x' < x$, and that $\theta(x, y) = true$ implies $\theta(x, y') = true$ for any $y' > y$. In other words, $\theta(x, y)$ is monotonically decreasing for x and increasing for y. Now we consider a general scheme of the itemset mining problems, to find the largest σ satisfying $\theta(\sigma, \kappa(\sigma))$. Note that $\kappa(\sigma)$ is monotonically decreasing for $1 \leq \sigma \leq m$, thus $\theta(\sigma, \kappa(\sigma))$ is monotonically decreasing for the same $1 \leq \sigma \leq m$. We call the largest σ *maximum frequency* for the threshold function, and denote it by σ_{max}.

For examples of such threshold functions, the function for top-k mining can be defined as $\theta(x, y) = true$ iff $y \geq k$. We then explore the maximum frequency σ_{max} for satisfying $\theta(\sigma, \kappa(\sigma))$. It means that the k-th most frequent itemsets have the frequency σ_{max}.

Here we define a threshold function for the LAMP condition as follows.

$$\theta(x, y) = true \ \text{ iff } \ f(x-1) > \frac{\alpha}{y} \tag{3}$$

As discussed in Section 3, we can confirm that this threshold function is decreasing for x and increasing for y. We then explore the maximum frequency σ_{max} for satisfying $\theta(\sigma, \kappa(\sigma))$[1]. Hereafter we assume that $\theta(x, y)$ satisfies the above condition.

[1]Our implementation uses $\kappa_c(\sigma)$ instead of $\kappa(\sigma)$ for the LAMP condition in equation (2). The number of closed itemsets are also monotonically decreasing for the frequency, so we can use $\kappa_c(\sigma)$ as well. This is discussed in Section 5.2

4.2 Support Increase Algorithm

For the computation of the maximum frequency for $\theta(\sigma, \kappa(\sigma))$, a natural way is to compute $\kappa(\sigma)$ by frequent itemset mining with the minimum support σ, for all possible candidates σ, one by one. This computation takes long time when σ is small since $\kappa(\sigma)$ is huge, thus we should compute $\theta(\sigma, \kappa(\sigma))$ in the decreasing order of σ. This is the basic scheme of the original LAMP [16] shown in the previous section. However, this needs long computation time since many frequent itemset mining processes are executed. In this paper, we propose a new algorithm that basically executes mining algorithm just once.

Suppose that we are given a function $\theta(\sigma, \kappa(\sigma))$. For a frequency σ, if we found some k and confirmed both $k \leq \kappa(\sigma)$ and $\theta(\sigma, k) = true$, then we get $\theta(\sigma, \kappa(\sigma)) = true$, since $\theta(x, y)$ is increasing for y. In such a case, we can see $\sigma_{max} \geq \sigma$, and in particular, $\sigma_{max} = \sigma$ if $\theta(\sigma + 1, \kappa(\sigma + 1)) = false$.

Suppose that we execute a backtracking algorithm for minimum support σ to check $\theta(\sigma, \kappa(\sigma))$, and during the mining process we found k frequent itemsets satisfying $\theta(\sigma, k)$. From the assumption of the function $\theta(x, y)$, we can then confirm that $\theta(\sigma, \kappa(\sigma)) = true$, thus we next. We are then motivated to re-execute the backtracking with $\sigma := \sigma + 1$ to check $\theta(\sigma + 1, \kappa(\sigma + 1))$. However, in the current execution, we already found possibly many itemsets of frequency $\sigma + 1$, and in the past search process, we never missed such itemsets. This implies that the execution until the current iteration can be skipped. We just need to remove all itemsets of frequency σ from the set of past solutions that have already been found, and can re-start the backtracking algorithm with $\sigma := \sigma + 1$. This implies that even if we start from very small σ having huge $\kappa(\sigma)$, we can increase it during the mining process for reducing the computation time.

A backtracking algorithm with this idea can be written as follows. Figure 4 illustrates the scheme of this algorithm. The algorithm start with $\sigma = 1$ and $\mathcal{S} = \emptyset$ where $\mathcal{S}$ is a program variable for storing a set of frequent itemsets we already found. Let $\mathcal{S}[\sigma]$ be the subset of itemsets in $\mathcal{S}$ whose frequency is σ.

```
ALGORITHM SupportIncrease ($X$)
global variable: $\sigma, \mathcal{S}$ (initialized $\sigma = 1$, $\mathcal{S} = \emptyset$)
1. $\mathcal{S} := \mathcal{S} \cup \{X\}$
2. if $\theta(\sigma, |\mathcal{S}|) = true$ then $\mathcal{S} := \mathcal{S} \setminus \mathcal{S}[\sigma]$, $\sigma := \sigma + 1$; go to 2
3. for each item $e > tail(X)$,
     if $frq(X \cup e) \geq \sigma$ then call SupportIncrease ($X \cup e$)
```

Let σ^* and $\mathcal{S}^*$ be the resulted values of σ and $\mathcal{S}$ after the execution of the algorithm.

Theorem 1. *There hold* $\sigma^* = \sigma_{max} + 1$ *and* $|\mathcal{S}^*| = \kappa(\sigma_{max} + 1)$.

Proof. Since the algorithm never decreases σ, the itemsets of frequencies no less than σ^* cannot be missed, and are included in $\mathcal{S}^*$. Since no itemset of frequency less than σ^* is in $\mathcal{S}$, we have $|\mathcal{S}^*| = \kappa(\sigma^*)$. We then prove $\sigma^* = \sigma_{max} + 1$ by contradiction.

Suppose that $\theta(\sigma^*, \kappa(\sigma^*)) = true$ holds. Let us consider the iteration in which the last itemset is inserted to $\mathcal{S}$. Since $\theta(\sigma^*, \kappa(\sigma^*)) = true$ holds, the latter part of step 2 is not executed in this iteration, otherwise σ is increased so that $\theta(\sigma^*, \kappa(\sigma^*)) = false$ holds. Thus, we have $\sigma = \sigma^*$ in the iteration. After step 1, there is no more itemset of frequency no less than σ^* that has not been found, thus we have $|\mathcal{S}| = \kappa(\sigma)$. This leads that $\theta(\sigma^*, \kappa(\sigma^*)) = false$, and contradicts to the assumption.

We next suppose that $\theta(\sigma^* - 1, \kappa(\sigma^* - 1)) = false$ holds. Let us consider the iteration in which σ is increased to σ^*. In the iteration, the latter part of step 2 is executed, and thus we have $\theta(\sigma^* - 1, |\mathcal{S}|) = true$. Since $|\mathcal{S}|$ is always no greater than $\kappa(\sigma)$, it implies that $\theta(\sigma^* - 1, k) = true$ holds for some $k \leq \kappa(\sigma^* - 1)$. This contradicts to the assumption. □

The algorithm terminates in short time if $|\mathcal{S}[\sigma]|$ is relatively small compared to $|\mathcal{S}|$ on average. Particularly, if $|\mathcal{S}[\sigma]| \leq q$ always holds, we can bound the number of iterations by $q \cdot \sigma_{max} + |\mathcal{S}^*|$, since the algorithm removes at most $q \cdot \sigma_{max}$ itemsets from $\mathcal{S}$. In the real-world data, we can naturally expect that $|\mathcal{S}[\sigma]|$ is much smaller than $|\mathcal{S}|$. In fact, we could confirm it in our computational experiments.

The computation of $|\mathcal{S}[\sigma]|$ is done efficiently by using a heap that extracts the minimum frequency itemset from $\mathcal{S}$. The computation then takes $O(|\mathcal{S}[\sigma]| \log |\mathcal{S}|)$ time. This computation time should be short than the computation time of an iteration, thus the total computation is expected not to increase much. However, if $|\mathcal{S}|$ is large on average, or $\kappa(\sigma^*)$ is very large, the algorithm may take very long time. In the next section, we focus on such cases, and propose an efficient method for reducing the computation time.

5 Fast Implementation

The bottleneck of the computation of the algorithm in the previous section comes from the large size of the heap for $\mathcal{S}$. This effects not only the computation time but also the memory usage. We here propose to use histogram counters instead of the heap. The histogram counters $c[\sigma']$ are prepared for each frequency σ', and keep the number of itemsets in $\mathcal{S}$ whose frequency is σ'. By using the histogram counters, we can compute $|\mathcal{S}|$ by summing up the counters for all $\sigma' \geq \sigma$. Hence, we do not have to use the heap for storing the itemsets. The point is that we can compute $|\mathcal{S} \setminus \mathcal{S}[\sigma]|$ by summing up the counters for all $\sigma' \geq \sigma + 1$. This implies that we have to do nothing about the update of $\mathcal{S}$ in step 2, thus computation will be smart. Another important point is the memory usage. The number of the histogram counters can be bounded by the number of transactions, thus it is only a linear factor to the input data size.

The use of histogram counters gives us one more advantage; we can use the equi-support technique. In each iteration with equi-support, we find many frequent itemsets, $2^{|Eq|}$ itemsets in exact, with the same frequency, at once. We can increase the counter for these in one step by adding $2^{|Eq|}$ to the counter. This saves the computation time of $2^{|Eq|}-1$ iterations, thus the equi-support technique drastically shortens the total computation time. The algorithm is then written as follows.

```
ALGORITHM **EquiSupportIncrease** ($X$)
global variable: $\sigma, c[]$ (initialized $\sigma = 1$, $c[i] = 0$ for each $i$)
1. $c[frq(X)] := c[frq(X)] + 2^{|Eq(X)|}$
2. **if** $\theta(\sigma, \sum_{i=\sigma}^{|\mathcal{D}|} c[i]) = true$ **then** $\sigma := \sigma + 1$; **go to** 2
3. **for** each item $e > tail(X)$,
   **if** $frq(X \cup e) \geq \sigma$ **then call EquiSupportIncrease** ($X \cup e$)
```

Theorem 2. *After the execution of algorithm EquiSupportIncrease, $\sigma^* = \sigma_{min} + 1$ holds, and $c[\sigma'] = \kappa(\sigma') - \kappa(\sigma' + 1)$ holds for all $\sigma' \geq \sigma^*$.*

Proof. We assume that the variable $\mathcal{S}$ is simultaneously computed as algorithm **SupportIncrease** during the execution of **EquiSupportIncrease**. Instead of insertion of itemsets to $\mathcal{S}$, EquiSupportIncrease increases a counter of the frequency of the itemsets. Thus, in any iteration, $c[\sigma'] = |\{Y \in \mathcal{S} | frq(Y) = \sigma'\}|$ holds. We then have $\sum_{i=\sigma}^{|\mathcal{D}|} c[i] = |\mathcal{S}|$, thus $\theta(\sigma, \sum_{i=\sigma}^{|\mathcal{D}|} c[i]) = true$ if and only if $\theta(\sigma, |\mathcal{S}|) = true$. This implies that the computation of σ in EquiSupportIncrease is the same as SupportIncrease, thus the statement holds. □

Theorem 3. *Let P be the itemset enumerated by* **EquiSupportIncrease***. The number of iterations needed to enumerate P by* **EquiSupportIncrease** *is equal to that by backtracking algorithms with equi-support technique.* □

5.1 Calculating Family Size of the LAMP

Theorem 1 states that $|\mathcal{S}^*| = \kappa(\sigma_{max} + 1)$, namely, the histogram counters holds $\kappa(\sigma_{max} + 1)$ after the execution of the proposed algorithm. However, our final purpose of the LAMP is to know $\kappa(\sigma_{max})$, which is the hypothesis family size for the P-value correction. An easy way to know $\kappa(\sigma_{max})$ is calling LCM algorithm once again with σ_{max}. This is not so bad since calculating $\kappa(\sigma_{max})$ is less time-consuming than finding σ_{max}, however, we can avoid such two-pass executions if we maintain not only $c[\sigma]$ but also $c[\sigma - 1]$ during the backtracking procedure. This can be done by a very small modification as shown below.

```
ALGORITHM **EquiSupportIncreaseLAMP** ($X$)
global variable: $\sigma, c[]$ (initialized $\sigma = 1$, $c[i] = 0$ for each $i$)
1. $c[frq(X)] := c[frq(X)] + 2^{|Eq(X)|}$
2. if $\theta(\sigma, \sum_{i=\sigma}^{|\mathcal{D}|} c[i]) = true$ then $\sigma := \sigma + 1$; go to 2
3. for each item $e > tail(X)$,
   if $frq(X \cup e) \geq \sigma - 1$ then call EquiSupportIncreaseLAMP ($X \cup e$)
```

In this modification, the total number of backtracking will be a little increase because the condition of the recursive call in Step 3 is relaxed to one smaller frequency. This overhead is relatively small in the total computation time, and it is a reasonable cost for computing $\kappa(\sigma_{max})$ correctly.

5.2 Generalization to Other Patterns

Many patterns have been considered in the past researches of pattern mining. Our algorithm works on many of these patterns. The requirement is that the backtracking algorithm does work, i.e., the set of frequent patterns satisfies some monotone properties. For example, closed itemset mining accepts our algorithm, and also maximal frequent itemset mining also does. Closed itemsets have the anti-monotone property, and maximal frequent itemsets can be enumerated by backtracking. They can be solved by our implementation. Sequence pattern mining [21], frequent tree mining [4, 5], frequent geograph mining [3], maximal motif mining [2], frequent graph mining [12], and other basic patterns also satisfy monotone property, so that we can construct a search tree in which parents have frequencies no smaller than their children, thereby backtracking works. They accepts the counter implementation when the maximum possible frequency is not huge, and also Equi-support technique when it works. Our basic scheme of the algorithm is quite strong so that we can use it in many kinds of pattern mining problems.

6 Computational Experiments

We implemented our new LAMP algorithm by modifying LCM ver. 5 that is available on the author's website [17]. This is the latest version of LCM algorithm that won in FIMI04 competition of fast pattern mining implementations [8]. The fundamental issues of the implementation is described in [18]. We note that the modification is quite small such that we added/modified only up to 30 lines of C codes. We could not observe any relatively large difference of computational performance after the modification of the LCM. For the comparison, we also evaluated the original version of LAMP [15]. The original LAMP repeatedly calls the same LCM ver. 5, as shown in Section 3 of this paper.

Table 1 presents the specifications of the database instances used in our experiments. “yeast” and “human cancer” are the real gene databases, which are

Table 1: Specifications of databases

database	n	m	Bonferroni (family size)		
			max-arity: 2	max-arity: 3	max-arity: 4
yeast	102	6074	5253	176953	4426528
breast cancer	397	12773	79003	10428793	1029883108
mushroom(3-119)	117	8124	6903	267033	7680738
T10I4D100K	870	100000	378885	109751225	23816205920
BMS-WebView-1	497	59602	123753	20460993	2532110133
BMS-WebView-2	3340	77512	5579470	6209953450	5.182×10^{12}
BMS-POS	1657	515597	1373653	758258113	3.137×10^{11}

Table 2: Experimental Results of new and original LAMP

database	p	σ_{max}	$\kappa_c(\sigma_{max})$ (family size)	new LAMP time(sec)	orig. LAMP time(sec)
yeast	530	4	303	0.005	0.463
breast cancer	1129	8	3750336	36.538	86.315
mushroom(3-119)	3916	20	98723	0.740	141.327
T10I4D100K(p-50)	50000	21	107080	3.092	799.738
T10I4D100K(p-10)	10000	7	483300	5.714	820.756
BMS-WebView-1(p-50)	29801	22	170660	12.349	122.303
BMS-WebView-1(p-10)	5960	8	959435	33.351	248.055
BMS-WebView-2(p-50)	38756	22	209016	1.813	229.406
BMS-WebView-2(p-10)	7751	8	665411	5.655	246.278
BMS-POS(p-50)	257799	31	74373743	580.321	78801.858
BMS-POS(p-10)	51560	11	702878145	4513.052	50883.609

Intel Core i7-3930K 3.2GHz, 64GB Mem, 12MB Cache, OpenSuSE 12.1

also used in the original LAMP paper [16]. The others are well known benchmark datasets of FIMI competition and KDD CUP 2000, available from the FIMI repository [8]. The columns n and m show the numbers of items and transactions, respectively. Here we show the hypothesis family sizes of traditional Bonferroni correction if we limit the maximum arity (the number of items in a combination) up to 2, 3, and 4. We can see that the family size grows exponentially to the arity, and that it seems too large for meaningful knowledge discovery in practical size of databases.

Table 2 shows our experimental results of the new and original versions of LAMP. The columns p, σ_{max} and $\kappa_c(\sigma_{max})$ indicate the number of positive transactions, the maximum frequency for the LAMP condition, and the number of frequent closed itemsets (hypothesis family size by LAMP), respectively. Here we used the significance threshold $\alpha = 0.05$. Note that the datasets "yeast" and "human cancer" have a positive-negative classification for each transaction. "mushroom" does not have such information but it has the two specific items which mean poisonous or edible mushroom, so we define the poisonous one as positive, and the rest of 117 items are assessed as the combinatorial hypotheses. For the other datasets, we did not specify a particular classification item, but we assumed the two cases that the positive transactions share 50% and 10% in the whole data. The experimental results show that the new LAMP is much faster than the original LAMP, as much as 10 to 100 times in many cases, and it can work well for practical size of databases with hundreds of items and thousands of transactions. We can also see that the family sizes given by LAMP are often smaller than ones by the traditional Bonferroni with the max-arity up to only 3. This is very powerful in practical applications because the LAMP has no arity limit up to n, and this correction still guarantees the family-wise error rate bounded by α.

7 Conclusion

In this paper, we proposed a fast itemset enumeration algorithm to find the frequency threshold satisfying the LAMP condition. We developed a general scheme to explore the maximum frequency satisfying a monotonic threshold function. We successfully applied this new scheme to the LAMP condition. The procedure is implemented into the newest LCM program. Our experimental results show that the proposed method is much (10 to 100 times) faster than the original LAMP and that it can work well for practical size of experiment databases. The new enumeration algorithm solved the bottleneck of the LAMP for practical applications, and useful for various areas of experimental sciences.

We may have several kinds of future work. As well as multiple-testing correction, computing the P-values for a particular hypothesis is also time-consuming procedure. It will be useful if we can efficiently compute the P-values for many combinatorial hypotheses and can discover the best or top-k significant one. In this paper, we considered Fisher's exact test, however, there are some other types of the P-value calculation, such as χ-squared test and Mann-Whitney test, and we may consider different statistical models. In addition, here we assumed only the binary-valued databases, but extension to non binary-valued databases is also interesting problem. As described in Section 5.2, our basic scheme of the algorithm is quite strong so that we can use it in many kinds of pattern mining problems. Anyway, our result demonstrated that the state-of-the-art enumeration techniques of pattern mining can be a useful means to many kinds of statistical problems.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Vol. 22(2) of SIGMOD Record*, pages 207–216, 1993.

[2] Hiroki Arimura and Takeaki Uno. A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. In Xiaotie Deng and Ding-Zhu Du, editors, *Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 724–737. Springer Berlin Heidelberg, 2005.

[3] Hiroki Arimura, Takeaki Uno, and Shinichi Shimozono. Time and space efficient discovery of maximal geometric graphs. In Vincent Corruble, Masayuki Takeda, and Einoshin Suzuki, editors, *Discovery Science*, volume 4755 of *Lecture Notes in Computer Science*, pages 42–55. Springer Berlin Heidelberg, 2007.

[4] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. *IEICE Trans. on Information and Systems*, E87-D(12):2754–2763, 2004.

[5] Tatsuya Asai, Hiroki Arimura, Takeaki Uno, and Shin-ichi Nakano. Discovering frequent substructures in large unordered trees. In Gunter Grieser, Yuzuru Tanaka, and Akihiro Yamamoto, editors, *Discovery Science*, volume 2843 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin Heidelberg, 2003.

[6] C. Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze (Libreria Internazionale Seeber, Florence, Italy)*, 8:3–62, 1936.

[7] B. Goethals. Survey on frequent pattern mining, 2003. http://www.cs.helsinki.fi/u/goethals /publications/survey.ps.

[8] B. Goethals and M. J. Zaki. Frequent itemset mining dataset repository, 2003. Frequent Itemset Mining Implementations (FIMI'03), http://fimi.cs.helsinki.fi/.

[9] Nature Publishing Group. Nature guide to authors: Statistical checklist. http://www.nature.com/nature/authors/gta/Statistical_checklist.doc.

[10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *In Proc of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, 2000.

[11] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[12] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In DjamelA. Zighed, Jan Komorowski, and Jan Zytkow, editors, *Principles of Data Mining*

and Knowledge Discovery, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23. Springer Berlin Heidelberg, 2000.

[13] Keisuke Okita, Tomoko Ichisaka, and Shinya Yamanaka. Generation of germline-competent induced pluripotent stem cells. *Nature*, 448:313–317, 2007.

[14] Mark J. van der Laan Sandrine Dudoit. *Multiple Testing Procedures with Applications to Genomics*. Springer New York, 2008.

[15] Aika Terada, Mariko Okada-Hatakeyama, Koji Tsuda, and Jun Sese. LAMP limitless-arity multiple-testing procedure, 2013. http://a-terada.github.io/lamp/.

[16] Aika Terada, Mariko Okada-Hatakeyama, Koji Tsuda, and Jun Sese. Statistical significance of combinatorial regulations. *Proceedings of National Academy of Sciences of United States of America*, 110(32):12996–13001, 2013.

[17] T. Uno. Program codes of takeaki uno. http://research.nii.ac.jp/˜uno/codes.htm.

[18] T. Uno, M. Kiyomi, and H. Arimura. LCM ver.2: efficient mining algorithms for frequent/closed/maximal itemsets. In *Proc. IEEE ICDM'04 Workshop FIMI'04 (International Conference on Data Mining, Frequent Itemset Mining Implementations)*, 2004.

[19] T. Uno, M. Kiyomi, and H. Arimura. LCM ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proc. Open Source Data Mining Workshop on Frequent Pattern Mining Implementations 2005*, 2005.

[20] T. Uno, Y. Uchida, T. Asai, and H. Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *Proc. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003. http://fimi.cs.helsinki.fi/src/.

[21] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proc of 4th IEEE International Conference on Data Mining (ICDM 2004)*, pages 79–90, 2007.

[22] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(2):372–390, 2000.