Practical Techniques for Speeding Up Enumeration Algorithms for Frequent Itemset Mining Problems

Takeaki Uno[1] **Tatsuya Asai**[2] **Hiroki Arimura**[2] **Yuzo Uchida**[2]
[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, JAPAN
e-mail:uno@nii.jp
[2] Department of Informatics, Kyushu University, 6-10-1 Hakozaki, Fukuoka 812-0053, JAPAN
e-mail:{t-asai,y-uchida,arim}@i.kyushu-u.ac.jp

Abstract: In this paper, we address practical techniques for efficiently mining all frequent item sets, frequent closed item sets, and maximal frequent item sets, respectively, from transaction databases. We modify the algorithm for enumerating maximal bipartite cliques proposed in SIGAL89 [9, 10], and propose an algorithm for frequent closed item set enumeration. The computation time of the algorithm is theoretically proved to be linear in the number of output, which is not proved for existing algorithms. We further modify the algorithm for practical computation, and for enumerating all frequent item sets, and maximal frequent item sets. By computational experiments, which is done by a competition of frequent set mining algorithms, we showed our approach is quite efficient, especially for realworld problems witch are said to be difficult.

1 Introduction

Frequent item set mining is one of the most fundamental problems in data mining and has many applications such as association rule mining [1], inductive databases [6], and query expansion [8]. Let E be an item set, composed items $1, ..., n$. $\mathcal{T}$ is a set of transactions over E, i.e., each $T \in \mathcal{T}$ is composed of items of E. For an item set $K \subseteq E$, let $\mathcal{T}(K) = \{T | T \in \mathcal{T}, S \subseteq t\}$. Each transaction of $\mathcal{T}(K)$ is called *occurrence* of K, and $\mathcal{T}(K)$ is called the *occurrence set* of K. For a given constant α, an item set K is called *frequent* if $|\mathcal{T}(K)| \geq \alpha$. We call α *support*. If a frequent item set is included in no other frequent item set, the item set is said to be *maximal*. For a transaction set $\mathcal{S} \subseteq \mathcal{T}$, let $\mathcal{I}(\mathcal{S}) = \bigcap_{T \in \mathcal{S}} T$. If an item set K satisfies $\mathcal{I}(\mathcal{T}(K)) = K$, K is called *closed item set*.

In this paper, we consider problems of enumerating all frequent item sets, frequent closed item sets, and maximal frequent item sets for given transactions. As shown in [3], frequent closed item set mining problems can be considered as a problem of enumerating maximal bipartite cliques. Given a collections of tuples of items as a database, we can define the associated bipartite graph consisting of two disjoint sets of nodes, one for items and another for tuples. The bipartite graph has an edge from an item to a tuple iff the item appears in the tuple. In the graph, an item set defines a set of tuples as its occurrences, and a set of tuples defines an item set as their maximally common substructure. Particularly, the closed item sets are equivalent to maximal bipartite cliques of a bipartite graph since a closed item set is the unique maximum one among those item sets that have the same set of occurrences.

According to the above observation, we can enumerate all closed item sets by using enumeration algorithms for maximal bipartite cliques. However, the algorithm outputs many infrequent closed item sets, thus we need some modifications. In SIGAL 89, one of the authors proposed a practical fast algorithm for enumerating maximal bipartite cliques, whose computation time is theoretically bounded by $O(\Delta^3)$ for each maximal bipartite clique, and by $O(\bar{\Delta}^2)$ experimentally, where Δ and $\bar{\Delta}$ are the maximum degree and the average degree of the input graph, respectively [9, 10]. Roughly speaking, the algorithm outputs bipartite cliques in the decreasing order of the number of vertices included in $\mathcal{T}$, in its recursion. In detail, if an iteration outputs a bipartite clique including x vertices of $\mathcal{T}$, any its descendant iteration outputs a clique including vertices of $\mathcal{T}$ less than x. Hence, when we enumerate closed item sets by this algorithm, if an iteration outputs an infrequent closed item set, then any descendant of the iteration always outputs infrequent closed item set. Thus, we can omit such iterations and can enumerate frequent closed item sets in linear time of the number of frequent closed item sets.

We further use several techniques for reducing the practical computation time, occurrence deliver, partial adjacent matrix, hybrid, and rightmost sweep. Occurrence deliver is a way to reduce the time for computing the frequency, partial adjacent matrix reduces the computational costs of adjacency matrix, hybrid switches the strategy of computing occurrence sets, and rightmost sweep reduces the memory complexity. By using these techniques, we reduce the practical computation time of the algorithm.

Algorithms for mining all frequent item sets, and maximal frequent item sets are obtained with slight modifications to the algorithm. Since any maximal frequent item set is a closed item set, we can enumerate all maximal frequent item sets by enumerating closed item sets and check the maximality of them. To enumerate all frequent item sets, backtrack algorithms are often used. The computation time of backtrack algorithms are linear in the number of frequent item sets. However, they take much computation time for checking the frequency, and this is a bottle neck part. Closed item sets classifies the item sets into equivalence classes such that each item set of an equivalence class has the same frequency. Thus, we use the frequent closed item set enumeration, and enumerate all the frequent item sets in the equivalence classes induced by the frequent closed item sets. This needs fewer frequency checks than backtracking algorithms, and we can reduce the practical computation time.

We show the efficiency of the algorithm by computational experiments done in a competition of frequent item set mining algorithms. We categorize the instances into four groups by the number of frequent closed item sets and supports. We consider the performance and advantages of our techniques via the results.

In the following sections, we explain our algorithms and practical techniques. We first explain the algorithm proposed in [9, 10] in the terms of frequent item sets in Section 2. Then, we propose practical techniques for speeding up in Section 3. In Section 4 and 5, we present modifications for all frequent item sets and maximal frequent item sets enumeration. Section 6 shows the results of experiments done in FIMI'03, and discuss the efficiency of our algorithms and techniques. Finally, Section 7 concludes this paper.

2 Enumerating Frequent Closed Item Sets

In this section, we explain a rough sketch of the algorithm for enumerating maximal bipartite cliques proposed in [9, 10]. Here we use the terms of frequent item set mining.

Let $\mathcal{C}$ be the set of frequent closed item sets. For $K \in \mathcal{C}$, let the parent index, denoted by $i(K)$, be the minimum item such that $\mathcal{T}(K) = \mathcal{T}(K \cap \{1, ..., i\})$. We define the parent of K by $\mathcal{I}(\mathcal{T}(K \cap \{1, ..., i(K) - 1\}))$. Since $\mathcal{T}(K) \subset \mathcal{T}(K \cap \{1, ..., i(K) - 1\})$, the frequency of the parent is larger than a child. Thus, the graph representation of the parent-child relationship forms a tree, whose root is $\emptyset$. We call the tree *enumeration tree*. We can find all frequent closed item set by traversing the enumeration tree.

To traverse the enumeration tree, we need to find children of a closed item set. For an item set K and an index i, let $K[i] = \mathcal{I}(\mathcal{T}(K \cup \{i\}))$. Here we state the following lemma.

Lemma 1 *K' is a child of $K \in \mathcal{C}$ if and only if (1) $K[i] = K'$, (2) $i > i(K)$, and (3) $K[i] \setminus K$ includes no item less than i.*

Proof: Suppose that K' is a child of K. Then, $K = \mathcal{I}(\mathcal{T}(K' \cap \{1, ..., i(K') - 1\}))$. This implies that $i(K) < i(K')$ and $K[i] = K'$. Hence (1) and (2) hold. Since $\mathcal{T}(K') \subset \mathcal{T}(K)$, we have $K \subset K'$. This together with $K = \mathcal{I}(\mathcal{T}(K' \cap \{1, ..., i(K') - 1\}))$ implies that $K \cap \{1, ..., i(K') - 1\} = K' \cap \{1, ..., i(K') - 1\}$. Thus (3) holds.

Suppose that (1), (2) and (3) hold. Then, the parent index of $K[i]$ is i. From (3), $K[i] \cap \{1, ..., i - 1\} = K \cap \{1, ..., i - 1\}$, Hence, the parent of $K[i]$ is

$$\mathcal{I}(\mathcal{T}(K[i] \cap \{1, ..., i(K[i]) - 1\})) = \mathcal{I}(\mathcal{T}(K \cap \{1, ..., i - 1\})).$$

This together with (2) means that the parent of $K[i]$ is K. ∎

According to this lemma, we can generate any child of K by computing $K[i]$ for each $i > i(K)$. We describe the algorithm **LCMclosed** as follows.

```
LCMclosed ($K$:frequent closed item set)
Output $K$
For each $i > i(K)$ do
  If $K[i]$ is a child of $K$, and $K[i]$ is frequent then LCMclosed ($K[i]$)
```

3 Reducing Practical Computation Time

In this section, we explain the following techniques for reducing the practical computation time:

· occurrence deliver
· checking closedness with partial adjacency matrix
· diffsets and switching the strategy (hybrid)
· rightmost sweep.

We first explain occurrence deliver. This is for constructing $\mathcal{T}(K[i])$ for each $i > i(K)$ quickly. Suppose that we have computed $\mathcal{T}(K)$. Then, a simple way for computing $\mathcal{T}(K[i])$ is to find all occurrences of K including i. This takes $O(\mathcal{T}(K))$ time for each i. Instead of this, we trace each occurrence T of K, and insert T to $Occ[i]$ for each items $i \in T$ larger than $i(K)$. After tracing all occurrences of K, $Occ[i] = \mathcal{T}(K[i])$ holds for any i. In this way, we can compute $\mathcal{T}(K[i])$ in $O(|\mathcal{T}(K[i])|)$ time on average. We describe occurrence deliver in the following.

```
Occurrence_deliver ($\mathcal{T}(K)$:occurrence set, $i(K)$:parent index)
1. Set $Occ[i] := \emptyset$ for each $i > i(K)$
2. For each $T \in \mathcal{T}(K)$ do
3.   For each $i \in T, i > i(K)$ do
4.     Insert $T$ to $Occ[i]$
```

We note that the initializing of Occ at line 1 can be omitted by clearing all non-empty Occ's after each execution of **Occurrence_deliver**. Thus, line 1 does not take $O(|E|)$ time, and the time complexity of **Occurrence_deliver** is $O(\sum_{i>i(K)} |\mathcal{T}(K[i])|)$.

We second explain checking closedness with partial adjacency matrix. This technique is for checking whether $K[i] \cap \{1, ..., i-1\} = K \cap \{1, ..., i-1\}$ or not. This condition can be checked by executing occurrence deliver for items less than $i(K)$, but we can do better. $K[i] \cap \{1, ..., i-1\} = K \cap \{1, ..., i-1\}$ holds if and only if for any $j \in \{1, ..., i-1\} \setminus K$, some occurrences of K does not include j. We check this by finding occurrences of $K[i]$ not including j. The worst case time complexity of this check is as same as that by occurrence deliver, however we can terminate the check for j when we find an occurrence not including j. In particular, we have to check this for j included in $T^* \setminus K$ where T^* is of the minimum size in $\mathcal{T}(K[i])$. Hence, the experimental computation time is faster then occurrence deliver.

By using adjacency matrix, this check can be done quickly, however adjacency matrix takes much cost, both computation time and memory for large but sparse database. Hence, we use the rows of the adjacency matrix with respect to transactions with larger sizes. By this, we can reduce the computation time but do not take much cost for handling them. We describe the algorithm as follows.

```
LCMclosed_check ($K, \mathcal{T}(K[i]), i$)
$T^*$ := transaction of $\mathcal{T}(K[i])$ with the fewest items
For each $j \in T^* \setminus K, j < i$ do
```

```
3.   For each $T \in \mathcal{T}(K[i]) \setminus \{T^*\}$
4.     If $j \notin T'$ then go to 7.
5.   End for
6.   Return false
7. End for
8. Return true
```

We next explain hybrid. In the execution of the algorithm, we often have that $|\mathcal{T}(K[i])|$ is closed to $\mathcal{T}(K)$ for each child $K[i]$. In such cases, it is better that maintaining $\mathcal{T}(K) \setminus \mathcal{T}(K \cup \{i\})$ for all i in each iteration, and evaluating the frequency of $K[i]$ by $|\mathcal{T}(K)| - |\mathcal{T}(K) \setminus \mathcal{T}(K[i])| = |\mathcal{T}(K)| - |\mathcal{T}(K) \setminus \mathcal{T}(K \cup \{i\})|$. The set $\mathcal{T}(K) \setminus \mathcal{T}(K \cup \{i\})$ is called *diffset*. Diffsets can be updated quickly when we add an item to K. This technique is proposed by Zaki [11]. We do not have to maintain diffsets for items i with $|\mathcal{T}(K[i])| < \alpha$, thus the practical computation time is fast if many frequent item sets includes the same items. Diffsets can be also used to check whether $K[i] \cap \{1, ..., i-1\} = K \cap \{1, ..., i-1\}$ or not.

Now we have two strategies to compute the occurrence sets. The performance of two strategies depend on the problems, thus we estimate the computation time of both strategies in each iteration, and choose the best in each iteration. Particularly, if we once choose diffsets, then we use diffsets in all its descendant iterations since the estimated computation time must be shorter than that of occurrence deliver for any its descendant.

We next explain rightmost sweep which is to reduce the memory complexity of occurrence deliver. Occurrence deliver needs memory for *Occ*, which is up to the size of $O(|E|)$, in each iteration. If we do not re-use the memory in the descendant iterations, we have to use $O(|E||K_{max}|)$ memory in total where K_{max} is the largest frequent item set. Rightmost sweep enables us to re-use the memory.

The improvement in rightmost sweep is to generate recursive calls with respect to $K[i]$ in the decreasing order of i. Any iteration inputting frequent item set K accesses $Occ[j]$ for only $j > i(K)$. Hence, even if we re-use *Occ* in the descendant iterations with respect to $K[i]$, $Occ[j]$'s for $j < i$ are preserved. Thus, we do not need to re-compute $\mathcal{T}(K[j])$ for $j < i$.

4 Enumerating Frequent Item Sets

In this section, we explain our algorithm for enumerating all frequent item sets. To enumerate quickly, we use the following properties. We omit the proofs.

Property 1 *For two item sets K and K' satisfying $K \subseteq K'$ and $\mathcal{T}(K') = \mathcal{T}(K)$, any set $R, K \subseteq R \subseteq K'$ satisfies $\mathcal{T}(K) = \mathcal{T}(R) = \mathcal{T}(K')$. In particular, if and only if K is frequent, then R is frequent.* ∎

Property 2 *For two item sets K and K' satisfying $\mathcal{T}(K') = \mathcal{T}(K)$, $\mathcal{T}(K \cup S) = \mathcal{T}(K' \cup S)$ holds for any S.* ∎

Consider the following backtracking algorithm for enumerating all frequent item sets. The algorithm adds an item to the current item set, and generates a recursive call if the obtained item set is frequent.

```
Backtrack ($K$:item set)
Output $K$
For each $i >$ the largest item of $K$ do
  If $K \cup \{i\}$ is frequent then Backtrack ($K \cup \{i\}$)
```

Let $S(K)$ be the set of items larger than the largest item of K and included in $\mathcal{I}(\mathcal{T}(K))$. Suppose that we are executing an iteration, and $S(K) \neq \emptyset$. Then, from the above properties, we can see that any $R, K \subseteq R \subseteq K \cup S(K)$ is frequent if and only if K is frequent. This condition also holds if we add items not included in $S(K)$ to K, which will be output by descendant iterations of this iteration. Thus, we are motivated to omit the iterations with respect to item sets $R, R \cap S(K) \neq \emptyset$, by outputting R in the iteration with respect to $R \setminus S(K)$. By this, we can reduce the computation time concerned with the frequency of R.

This technique can be applied recursively. For any $K', K' \supseteq K, K' \cap S(K) = \emptyset$, we have $\mathcal{T}(K') = \mathcal{T}(R) = \mathcal{T}(K' \cup S(K) \cup S(K'))$ holds for any $K' \subseteq R \subseteq K' \cup S(K) \cup S(K')$. Recursively applying this, we can group item sets with the same occurrence sets. We call this technique *hypercube decomposition*, because an equivalence class with respect to a closed item set is decomposed into hypercubes of 01 lattice. From these discussion, we obtain the following algorithm.

```
LCMfreq ($K, S$:item set)
1. $S := S \cup S(K)$
2. Output all item sets $R, K \subseteq R \subseteq K \cup S$
3. For each $i >$ the largest item of $K$ do
4.   If $K \cup \{i\}$ is frequent then  LCMfreq ($K \cup \{i\}, K'$)
```

5 Enumerating Maximal Frequent Item Sets

The main idea of enumerating maximal frequent item sets is very simple. Since any maximal frequent item set is a frequent closed item set, we enumerate frequent closed item sets and output only those being maximal frequent item sets. The check of the maximality can be done in short time. K is a maximal frequent item set if and only if $\mathcal{T}(K \cup \{j\})$ is infrequent for any j. We can check this condition by executing occurrence deliver or diffsets for all items.

6 Computational Experiments

In 2003, a competition FIMI'03 of frequent item set mining algorithms had held. About 20 algorithms had been submitted to the competition, and evaluated their performances with over 10 instances with some supports such that the algorithms terminate in practical time. In the following subsections, we take a general view of these algorithms and data sets, and show the results.

6.1 Techniques on Other Algorithms

First of all, we explain common structures of the other algorithms submitted to FIMI'03. We can see the papers with respect to these algorithms are available at the homepage of FIMI'03[1]. Some basic ideas of these algorithms can be also seen in [1, 2, 4, 5, 7, 12, 11]. Almost all algorithms are based on the enumeration of frequent item sets, such as backtrack algorithms or apriori algorithms. Apriori algorithms generate each frequent item set from smaller one level by level in the bottom-up way, and avoids duplications by storing all item sets the algorithm found. The advantage of the apriori algorithms is shorter computation time of checking the frequency. An item set is frequent only if all its subsets are frequent. Apriori algorithms have all frequent item sets with smaller sizes in memory, thus they can check the condition in shorter time. To maintain the input data and frequent item sets found, many of them use "FP-tree," also called "trie", a kind of suffix tree. By using this, we can insert/delete/search a set in short time. However, if the average sizes and the number of frequent item sets gets large, they take much computation time and memory to store and maintain them.

To enumerate frequent closed item sets, these algorithms find all frequent item sets, and output only them being closed item sets. In addition, some of the algorithms omit some recursive

[1] http://fimi.cs.helsinki.fi/index.html

Table 1: The datasets. AvTrSz means the average transaction size

Dataset	#items	#Trans	AvTrSz	#FI	#FCI	#MFI	Minsup (%)
BMS-Web-View1	497	59,602	2.51	3.9K–NA	3.9K–1241K	2.1K–129.4K	0.1–0.01
BMS-Web-View2	3,340	77,512	4.62	24K–9897K	23K–755K	3.9K–118K	0.1–0.01
BMS-POS	1,657	517,255	6.5	122K–33400K	122K–21885K	30K–4280K	0.1–0.01
T10I5N1KP5KC0.25D200K	1,000	200,000	10.0	15K–335K	14K–229K	7.9K–114K	0.5–0.025
T30I15N1KP5KC0.25D200K	1,000	200,000	30.0	-	-	-	2–0.5
pumsb	7,117	49,046	74.0	-	-	-	95–55
pumsb_star	7,117	49,046	50.0	-	-	-	50–5
mushroom	120	8,124	23.0	-	-	-	20–0.1
connect	130	67,577	43.0	-	-	-	95–10
chess	76	3196	37.0	-	-	-	90–20
kosarak	130	67,577	43.0	-	-	-	1–0.1
retail	16470	88,162	10.3	-	-	-	0.1–0.01
accidents	469	340,183	33.8	-	-	-	90–30

calls such that no descendant iteration outputs closed item sets, but they seem not to be output linear time. To enumerate maximal frequent item sets, they also enumerate frequent item sets but uses many bounding operations, which omit the iterations whose descendants output no maximal frequent item set. Actually, this bounding operations are very powerful. Our algorithm has no such bounding operation for maximal frequent item sets, hence is slow for problems of maximal frequent item sets.

To reduce the input, these algorithms use a technique of input reduction. For an item i, if the number of transactions including i is smaller than the support, no frequent item set includes i. The algorithms remove such items from the input database. Moreover, if some transactions are the same, the algorithms contract the transactions. In the case that the support is large, many items are removed by this operation, hence the input size becomes small.

In summary, the techniques of the other algorithm which our algorithms does not use are:

(1) apriori
(2) input reduction
(3) bounding operations for maximal frequent item sets
(4) FP-tree

In the below, we discuss the efficiency of these techniques and our techniques

6.2 Datasets and Methods

The datasets are shown in Table 1, which include: T10I5N1KP5KC0.25D200K and T30I15N1KP5KC0.25D200K from IBM Almaden Quest research group; chess, connect, mushroom, pumsb, pumsb_star from UCI ML repository[2] and PUMSB; BMS-WebView-1, BMS-WebView-2, BMS-POS from KDD-CUP 2000[3]. We show the profiles of these datasets in Table 6.2.

Here we consider two parameters as measures of difficulty of datasets: (a) the minimum support which the algorithms terminates in short time, and (b) difference between the number of frequent closed item sets and frequent item sets.

If the support is large, the computation time concerned with the frequency will be long. Thus, input reduction and diffset are expected to be efficient, and apriori type algorithm may have an advantage. Conversely, in the case that the support is small, occurrence deliver, and output linear time algorithms have an advantage. If the number of frequent item sets are much larger

[2] http://www.ics.uci.edu/ mlearn/MLRepository.html

[3] http://www.ecn.purdue.edu/KDDCUP/

Table 2: Four categories

	Support is large	Support is small
# frequent item sets >> # frequent closed item sets	pumsb, pumsb_star connect, chess	BMS-Web-View1, BMS-Web-View2 retail, mushroom
# frequent item sets is nearly equal to # frequent closed item sets	accidents	T10I5N1KP5KC0.25D200K, T30I15N1KP5KC0.25D200K BMS-POS, kosarak

than frequent closed item sets, output linear time algorithm and hypercube decomposition have an advantage. In Table 6.2, we categorize these datasets by these two parameters.

6.3 Results

In Table 6.3 and 6.3, we show the outline of the results of the experiments done in FIMI'03. We report the best algorithm for the problems of the above four categories, with setting supports to be small and large. If our algorithm performs the best in a category, we write "ours" on the corresponding cell. If our algorithm does not works well, we write "others". If the performances of our algorithms and of the other algorithms are almost equal, then we write "both". We note that we write "others" even if only one of the other algorithm performs better then ours and the rest does not.

Table 3: Results of all frequent item sets, and frequent closed item sets

#frequent item sets >> #frequent closed item sets	small support	setting support to small	ours
		setting support to large	ours
	large support	setting support to small	ours
		setting support to large	others
#frequent item sets is nearly equal to #frequent closed item sets	small support	setting support to small	both
		setting support to large	both
	large support	setting support to small	others
		setting support to large	others

Table 4: Results of maximal frequent item sets

#frequent item sets >> #frequent closed item sets	small support	setting support to small	ours
		setting support to large	others
	large support	setting support to small	others
		setting support to large	others
#frequent item sets is nearly equal to #frequent closed item sets	small support	setting support to small	others
		setting support to large	others
	large support	setting support to small	others
		setting support to large	others

First of all, our algorithms work well but not the best for all. Particularly, for the instances of maximal frequent item sets, our algorithm is slow. This is because we use no bounding operation designed for the maximal frequent item set enumeration.

For frequent item sets and closed item sets, our algorithms perform well. Especially, when we set support to be small and the number of frequent item sets is larger than that of frequent closed item sets, our algorithms got high marks. However, when support is large, sometimes our algorithms are slow. This is because we do not use input reductions. In the case that we set support to be small, the performance of occurrence deliver is good, and otherwise diffset is good. In the middle range, the hybrid performs well.

We saw no advantages of apriori algorithm and FP-tree from the view points of these categories, which is said to be quite efficient for frequent item set mining problems. For the problems which our algorithms do not perform well, input reduction seems to contribute so much to speeding up. In the other cases, it seems to be better to use depth-first enumeration algorithms.

7 Conclusion

In this paper, we propose several practical efficient algorithms for enumerating all frequent closed item sets, frequent closed item sets, and maximal frequent item sets, from large transaction databases. These algorithms are obtained by adding several practical techniques to a maximal bipartite clique enumeration algorithm. In theory, we demonstrate that our algorithm exactly enumerates the set of frequent closed item sets in polynomial time for each frequent closed item set. We reported the results of a competition of frequent item set mining algorithms FIMI'03, and showed that our practical techniques have a good performance for large realworld problems such as BMS-Web-View1,2, and retail, and those having several artificial structures such as instances of UCI-repositories.

Acknowledgment

We gratefully thank to Professor Ken Satoh of National Institute of Informatics. This research had been supported by group research fund of National Institute of Informatics, JAPAN.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *In Proceedings of VLDB '94,* pp. 487–499, 1994.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, "Fast Discovery of Association Rules," *In Advances in Knowledge Discovery and Data Mining,* MIT Press, pp. 307–328, 1996.

[3] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, "On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets," *STACS 2002*, pp. 133-141, 2002.

[4] D. Burdick, M. Calimlim, J. Gehrke, "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases," *ICDE 2001*, pp. 443-452, 2001.

[5] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," *SIGMOD Conference 2000*, pp. 1-12, 2000

[6] H. Mannila, H. Toivonen, "Multiple Uses of Frequent Sets and Condensed Representations," *KDD'96*, pp. 189–194, 1996.

[7] J. Pei, J. Han, R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, pp. 21-30, 2000.

[8] B. Possas, N. Ziviani, W. Meira Jr., B. A. Ribeiro-Neto, "Set-based model: a new approach for information retrieval," *SIGIR'02*, pp. 230-237, 2002.

[9] Takeaki Uno, "A Practical Fast Algorithm for Enumerating Cliques in Huge Bipartite Graphs and Its Implementation," 89th Special Interest Group of Algorithms, Information Processing Society Japan, 2003,

[10] Takeaki Uno, "Fast Algorithms for Enumerating Cliques in Huge Graphs," Research Group of Computation, IEICE, Kyoto University, pp.55-62, 2003

[11] M. J. Zaki, C. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemset Mining," *2nd SIAM International Conference on Data Mining (SDM'02)*, pp. 457-473, 2002.

[12] M. J. Zaki, "Scalable algorithms for association mining," *Knowledge and Data Engineering*, 12(2), pp. 372–390, 2000.