Updating a Closed Itemset Family Based on Inclusion Relations

Shoichi Nishimura[1], Keisuke Otaki[1], Madori Ikeda[1],
Ryo Yoshinaka[1], Akihiro Yamamoto[1], Takeaki Uno[2]

[1] Graduate School of Informatics, Kyoto University, Kyoto, Japan 606-8501
{nishimura, ootaki, m.ikeda}@iip.ist.i.kyoto-u.ac.jp, {ry, akihiro}@i.kyoto-u.ac.jp
[2] National Institute of Informatics, Tokyo, Japan 101-8403
uno@nii.ac.jp

Abstract. Developing methods for enumerating closed itemsets is a main subject in mining from transaction databases. Usually databases are assumed to be updated, and therefore in this paper we deal with the problem of updating a closed itemset family when new transactions are inserted to an original database. In such situation, already existing closed itemsets are still closed after the update, and therefore we have only to enumerate new closed itemsets appeared. We propose an algorithm for enumerating the new closed itemsets in linear time with respect to the number of them. More precisely, the time complexity is $O(\Delta^4)$ per a new closed itemset where Δ is the maximum degree of a bipartite graph representing the given transaction database, and the space complexity is linear with respect to the database size.

1 Introduction

Frequent itemset mining is an important problem in data mining because it provides many fundamental ideas towards other mining problems, e.g., association rule mining and sequence mining. In the problem, a *transaction database*, which is a family of subsets of *items*, and a *threshold* are given, and the goal is to find every subset of items, called *frequent itemsets*, whose occurrence is more frequent than the threshold. Many algorithms have been proposed for solving the problem. They can solve it quickly in many cases, but they cannot in such cases that the threshold is little, or that many transactions include little number of items because the number of frequent itemsets becomes so large. In order to deal with such cases, it is efficient to focus only on *closed itemsets*. Because, for a given transaction database, the number of closed itemsets is smaller than the number of all frequent itemsets, and the set of all frequent itemsets is obtained from the set of frequent closed itemsets [4]. Many fast algorithms for enumerating such closed itemsets have been proposed [4–9]. For example, LCM [6–8], enumerates closed itemsets in linear time with respect to the number of the itemsets by a depth-first search on a tree consisting of frequent and closed itemsets.

Transaction databases are assumed to be frequently updated and consequently becomes larger and larger gradually. In mining such a database, we have to deal with change of the set of closed itemsets of the target database corresponding to its update. If we tried to enumerate all closed itemsets for each update of the target database, it would take exponential time in total even by the fast algorithms. In this paper, we assume that update of a database is carried out only by inserting new transactions, and, under the assumption, we propose to enumerate only new closed itemsets emerging from the update. Our algorithm enumerates the new itemsets in linear time for the number of them and needs a computational space in direct proportion to the size of the target transaction database.

This paper is organized as follows: After reviewing frequent itemset mining briefly in the following section, we treat the case that exactly one new transaction is inserted into an original database in Section 3. We present some relations between the closed itemsets in the original database and those in the updated database, and then show our algorithm for enumerating new closed itemsets. In Section 4 we discuss on the case that several new transactions are simultaneously inserted. We show our experimental evaluations in Section 5 and present our conclusion in Section 6.

2 Frequent Closed Itemset Mining

We formalize frequent closed itemset mining.

Let $\mathcal{I}$ be a set of *items*. A *transaction* t is a subset of $\mathcal{I}$, and a *transaction database* $\mathcal{T} \subseteq 2^{\mathcal{I}}$ is a set of transactions. For convenience, we notate each transaction in $\mathcal{T}$ with an index as $t_1, t_2, ..., t_{\mathcal{T}}$. When an item i and a transaction t satisfy $i \in t$, we say that i occurs in t, and that t contains i. In the same

Table 1. An transaction database $\mathcal{T}$.

t_1	$\{i_1, i_2, i_6\}$
t_2	$\{i_1, i_3, i_5\}$
t_3	$\{i_1, i_5, i_6\}$
t_4	$\{i_1, i_3, i_4\}$

Table 2. An updated database $\mathcal{T}^*$.

t_1	$\{i_1, i_2, i_6\}$
t_2	$\{i_1, i_3, i_5\}$
t_3	$\{i_1, i_5, i_6\}$
t_4	$\{i_1, i_3, i_4\}$
t_5	$\{i_3, i_5, i_6\}$

manner, for an itemset $I \subseteq \mathcal{I}$ and a transaction t satisfying $I \subseteq t$, we say that I occurs in t, and t is called an *occurrence* of I. For a transaction database $\mathcal{T}$, we define the frequency of an itemset $I \subseteq \mathcal{I}$ as $|\{t \in \mathcal{T} | I \subseteq t\}|$ and indicate it by $freq(I)$. When a number θ called a *support* is given, an itemset I satisfying $freq(I) \geq \theta$ is called *frequent*. *Frequent itemset mining* is to find all frequent itemsets from a given transaction database for a given support.

Closed itemsets are formally defined by using a closure operator.

Definition 1 (Closure operator and closed itemsets). Let $\mathcal{T} \subseteq 2^{\mathcal{I}}$ be a transaction database. For a subset of transactions $T \subseteq \mathcal{T}$ and an itemset $I \subseteq \mathcal{I}$, functions f and g, and a *closure operator* h are defined as follows.

$$f(T) = \{i \in \mathcal{I} \mid \forall t \in T, t \ni i\},\ g(I) = \{t \in \mathcal{T} \mid \forall i \in I, i \in t\}, \text{ and } h = f \circ g.$$

Intuitively, $f(T)$ returns a set of items occurring all transactions in T, and $g(I)$ returns a set of transactions containing all items in I. Together with an operator $h = f \circ g$, an itemset $I \subseteq \mathcal{I}$ is called a *closed itemset* if and only if $h(I) = I$. The set of all closed itemsets on $\mathcal{T}$ is denoted by $C(\mathcal{T})$. The set of all frequent closed itemsets is a representative set of all frequent itemsets. It is proved that the set of all frequent itemsets is obtained from the set of frequent closed itemsets [4].

For example, let $I = \{i_1, i_2, \ldots, i_6\}$ and $\mathcal{T}$ be the transaction database depicted in Table 1. Then

$$C(\mathcal{T}) = \{\{i_1, i_2, i_6\}, \{i_1, i_3, i_5\}, \{i_1\}, \{i_1, i_5, i_6\}, \{i_1, i_5\}, \{i_1, i_6\}, \{i_1, i_3, i_4\}, \{i_1, i_3\}\}.$$

3 Inserting One Transaction into a Database

In this section, we describe our algorithm based on the assumption that one transaction t^* is inserted in update.

Let $\mathcal{I}$ be an original set of items and $\mathcal{T} \subseteq 2^{\mathcal{I}}$ be an original database. We assume that, in a step of insertion, $\mathcal{T}$ is turned into $\mathcal{T}^* = \mathcal{T} \cup \{t^*\}$ such that $t^* \notin \mathcal{T}$. For the new transaction t^*, we assume that $t^* \subseteq \mathcal{I}^*$, and that $\mathcal{I}^* \supseteq \mathcal{I}$. For this new database $\mathcal{T}^*$, we use f^*, g^*, and h^* for each of the functions f, g, and h of $\mathcal{T}$ for clarity. For example, let $\mathcal{T}^*$ be the database shown in Table 2 obtained by inserting $t_5 = \{i_3, i_5, i_6\}$ into the database $\mathcal{T}$ in Table 1. Then

$$\begin{aligned} C(\mathcal{T}^*) = \{&\{i_1, i_2, i_6\}, \{i_1, i_3, i_5\}, \{i_1\}, \{i_1, i_5, i_6\}, \{i_1, i_5\}, \{i_1, i_6\}, \{i_1, i_3, i_4\}, \{i_1, i_3\}, \\ &\{i_3, i_5, i_6\}, \{i_3, i_5\}, \{i_5, i_6\}, \{i_3\}, \{i_5\}, \{i_6\}\}. \end{aligned}$$

Lemma 1. For a original database $\mathcal{T}$ and an updated database $\mathcal{T}^* = \mathcal{T} \cup \{t^*\}$, $C(\mathcal{T}) \subseteq C(\mathcal{T}^*)$.

Proof. Let t^* be a new transaction. If $I \in C(\mathcal{T})$ safisfies $I \subseteq t^*$, then $g^*(I) = g(I) \cup \{t^*\}$. Thus $f^*(g^*(I)) = f^*(g(I) \cup \{t^*\})$, and $f^*(g(I) \cup \{t^*\}) = f^*(g(I)) \cap f^*(\{t^*\})$ from the definition of the function f^*. It is clear that $f^*(T) = f(T)$ for a set of transaction T satisfying $T \not\ni t^*$, so $f^*(g(I)) = f(g(I)) = I$ because $I \in C(\mathcal{T})$. It is also clear that $f^*(\{t^*\}) = t^*$. Then, $f^*(g^*(I)) = I \cap t^* = I$, so $h^*(I) = I$. On the other hand, if $I \in C(\mathcal{T})$ safisfies $I \not\subseteq t^*$, $g^*(I) = g(I)$. Thus, $f^*(g^*(I)) = f^*(g(I)) = f(g(I)) = I$, and $h^*(I) = I$. Therefore, any closed itemset $I \in C(\mathcal{T})$ satisfies $h^*(I) = I$. This means that every closed itemset $I \in C(\mathcal{T})$ is also an element of $C(\mathcal{T}^*)$, i.e., $C(\mathcal{T}) \subseteq C(\mathcal{T}^*)$. $\square$

We define $NC = C(\mathcal{T}^*) \setminus C(\mathcal{T})$ and call every element of NC a *new closed itemset*. In the example,

$$NC = \{\{i_3, i_5, i_6\}, \{i_3, i_5\}, \{i_5, i_6\}, \{i_3\}, \{i_5\}, \{i_6\}\}.$$

Lemma 2. For an itemset $I \subseteq \mathcal{I}^*$, $I \in NC$ iff $f^*(g^*(I)) = I$ and $f^*(g^*(I) \setminus \{t^*\}) \not\subseteq t^* \in g^*(I)$.

Table 3. New closed itemsets of $\mathcal{T}^*$.

I	$f^*(g^*(I))$	$f^*(g^*(I) \setminus \{t^*\})$
$\{i_3, i_5, i_6\}$	$\{i_3, i_5, i_6\}$	$\{i_1, i_2, i_3, i_4, i_5, i_6\}$
$\{i_3, i_5\}$	$\{i_3, i_5\}$	$\{i_1, i_3, i_5\}$
$\{i_5, i_6\}$	$\{i_5, i_6\}$	$\{i_1, i_5, i_6\}$
$\{i_3\}$	$\{i_3\}$	$\{i_1, i_3\}$
$\{i_5\}$	$\{i_5\}$	$\{i_1, i_5\}$
$\{i_6\}$	$\{i_6\}$	$\{i_1, i_6\}$

Table 4. Parents of Enumeration Trees from $\mathcal{D}$ to $\mathcal{D}^*$.

I	T	$T(\text{mink}(T) - 1)$	$P(I)$
$\{i_3, i_5, i_6\}$	$\{t_5\}$	—	—
$\{i_3, i_5\}$	$\{t_2, t_5\}$	$\{t_5\}$	$\{i_3, i_5, i_6\}$
$\{i_5, i_6\}$	$\{t_3, t_5\}$	$\{t_5\}$	$\{i_3, i_5, i_6\}$
$\{i_3\}$	$\{t_2, t_4, t_5\}$	$\{t_2, t_5\}$	$\{i_3, i_5\}$
$\{i_5\}$	$\{t_2, t_3, t_5\}$	$\{t_2, t_5\}$	$\{i_3, i_5\}$
$\{i_6\}$	$\{t_1, t_3, t_5\}$	$\{t_5\}$	$\{i_3, i_5, i_6\}$

Proof. At first, from the definition of NC,

$$I \in NC \text{ iff } f(g(I)) \neq I \text{ and } f^*(g^*(I)) = I.$$

Since for all $I \subseteq \mathcal{I}^*$ $g(I) = g^*(I) \setminus \{t^*\}$ and $f^*(g^*(I) \setminus \{t^*\}) = f(g^*(I) \setminus \{t^*\})$, it holds that

$$f(g(I)) = f^*(g^*(I) \setminus \{t^*\}). \tag{1}$$

Then, $f^*(g^*(I) \setminus \{t^*\}) \neq I$ and $f^*(g^*(I)) = I$ implies $t^* \in g^*(I)$. Therefore,

$$I \in NC \text{ iff } f(g(I)) \neq I \text{ and } f^*(g^*(I)) = I \text{ and } t^* \in g^*(I).$$

From (1), $f(g(I)) \neq I$ iff $f^*(g^*(I) \setminus \{t^*\}) \neq I$. Because of anti-monotonicity of the function f^*, $f^*(g^*(I) \setminus \{t^*\}) \supseteq f^*(g^*(I)) = I$ when $I \in C(\mathcal{T}^*)$. Thus,

$$I \in NC \text{ iff } f^*(g^*(I)) = I \text{ and } t^* \in g^*(I) \text{ and } f^*(g^*(I) \setminus \{t^*\}) \supsetneq I.$$

If $t^* \in g^*(I)$, $f^*(g^*(I)) = f^*(g^*(I) \setminus \{t^*\}) \cap t^*$. Therefore, when $f^*(g^*(I)) = I$ and $t^* \in g^*(I)$, $f^*(g^*(I) \setminus \{t^*\}) \supsetneq I$ iff $f^*(g^*(I) \setminus \{t^*\}) \supsetneq f^*(g^*(I) \setminus \{t^*\}) \cap t^*$. This is equals to $f^*(g^*(I) \setminus \{t^*\}) \supsetneq I$ iff $f^*(g^*(I) \setminus \{t^*\}) \not\subseteq t^*$. Consequently, we obtain

$$I \in NC \text{ iff } f^*(g^*(I)) = I \text{ and } f^*(g^*(I) \setminus \{t^*\}) \not\subseteq t^* \in g^*(I).$$

□

We show an example in Table 3, where all of them satisfy the condition above.

To give our enumerating method, we consider rooted trees consisting of all new closed itemsets. In the following, we call such trees *enumeration trees*. We can enumerate new closed itemsets without duplications by traversing the trees because they have no cycles. For enumeration trees, we first define the root, denoted by I_0, of those trees. Here we adopt t^* as I_0. Note that we need to take care of the situation that t^* is not a new closed itemset. If $t^* \notin NC$, it obviously holds that $NC = \emptyset$. This means that we can prepare a pre-procedure of enumerating of all new closed itemsets to check whether or not $t^* \notin NC$. If t^* is not a new closed itemset, our enumeration procedure halts. If it is a new one, then we enumerate all new closed itemsets by traversing our enumeration trees. In the following, without loss of generality, we assume that $t^* \in NC$.

Next, we define a parent of each node of our enumeration trees to construct such trees. For a set $T \subseteq \mathcal{T}$ of transactions, we indicate $(T \cap \{t_1, t_2, \ldots, t_k\}) \cup \{t^*\}$ by $T(k)$, and indicate the minimum integer k satisfying $f^*(T(k)) = f^*(T)$ by mink(T). For some new closed itemset $I \in NC$ except t^*, we define the parent $P(I)$ of such itemset I with $T = g^*(I)$ as follows:

$$P(I) = f^*(T(\text{mink}(T) - 1)).$$

Moreover if $P(I') = I$ for two closed itemset $I, I' \in NC$, we define that I' *is a child* of I. Our enumeration trees are constructed by defining edges which connect each new closed itemset, except for the root, with its parent. Because of both $P(I) \supsetneq I$ and $P(I) \in NC$, such definition can give *trees*, that is, they have no cycle. We show a small example in Figure 1.

What we need to define in the following is an efficient method to find the children of some itemset I In the following we formally describe our updating methods by traversing enumeration trees. For some itemset I, some transaction T, and integers $k = 1, 2, \ldots, |T|$, we define $I[k] = f^*(T \cup \{t_k\})$ and $T[k] = T \cup \{t_l \in g^*(I[k]) \mid l \geq k\}$. Then, I' is a child of I if and only if the following lemma holds.

Lemma 3. Let $I \in NC$ and $T = g^*(I)$. Another itemset $I' \subseteq \mathcal{I}^*$ is a child of I if and only if there exists some integer k satisfying all of the following five conditions.

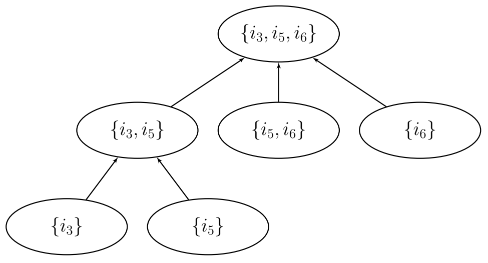

Fig. 1. An example of our enumeration tree consisting of new closed itemset.

Algorithm 1 Enumerating all new closed itemsets

```
 1: function ENUMERATE($\mathcal{T}^*, t^*$)
 2:     $I_0 \leftarrow t^*$
 3:     $T_0 \leftarrow g^*(I_0)$
 4:     if $f^*(T \setminus \{t^*\}) \not\subseteq t^*$ then
 5:         DFS($I_0, T_0, 0$)
 6: function DFS($I, T, mink$)
 7:     PRINT($I$)                                          ▷ $I$ is a new closed itemset
 8:     for $k = mink + 1$ to $|\mathcal{T}^*|$ do
 9:         $I' \leftarrow I[k]$
10:         $T' \leftarrow T[k]$
11:         if $t_k \notin T$ and $g^*(I') = T'$ and $f^*(T' \setminus \{t^*\}) \not\subseteq t^*$ then
12:             DFS($I', T', k$)
```

(a) $k > \mathrm{mink}(T)$,
(b) $t_k \notin T$,
(c) $I' = I[k]$,
(d) $g^*(I') = T[k]$, and
(e) $f^*(T[k] \setminus \{t^*\}) \not\subseteq I^*$.

Because of Lemma 3, for some integer k such that $\mathrm{mink}(T) + 1 \le k \le |\mathcal{T}|$, we can find all children of some itemset I only by checking whether or not some itemsets satisfy the conditions given in the lemma. By repeating such procedure at the root node, and by applying them recursively to children, we can traverse enumeration trees without storing them in main memory. We show our algorithm in Algorithm 1 of enumerating all new closed itemsets when a new transaction t^* is inserted.

Theorem 1. Let $\mathcal{T}^*$ be an updated database, $t^* \subseteq \mathcal{I}^*$ be a newly inserted transaction. The algorithm ENUMERATE$(\mathcal{T}^*, t^*)$ can enumerates all new closed itemsets without duplications correctly.

Proof. The function ENUMERATE first computes a set I_0 of items and $T_0 = g^*(I_0)$ which could become the root of an enumeration tree. By using the condition $f^*(T_0 \setminus \{t^*\}) \not\subseteq t^*$, we next to check whether or not $I_0 \in NC$. We discuss the following two cases: i) If $I_0 \notin NC$, then the function halts without outputs. Since $NC = \emptyset$, this result is correct. ii) If $I_0 \in NC$, it calls DFS$(I_0, T_0, 0)$ since $f^*(T_0 \setminus \{t^*\}) \not\subseteq I^*$. Then I_0 becomes the root of an enumeration tree. In the function DFS, for $k = mink + 1, mink + 2, \ldots, |\mathcal{T}|$, they check all itemsets satisfying all of $t_k \notin T$, $T[k] = g^*(I[k])$, $f^*(T[k] \setminus \{t^*\}) \not\subseteq t^*$. Because of Lemma 3, if $mink = \mathrm{mink}(T)$ then we can ensure that we check all children of I without duplications on the enumeration tree. For the root node, it holds that $0 = \mathrm{mink}(T_0)$. For the children of some index k, that is, for a call of DFS$(I[k], T[k], k)$, it holds that $k = \mathrm{mink}(T[k])$. Since it always hold that $mink = \mathrm{mink}(T)$, consequently this function DFS works correctly. □

We discuss the time and space complexities of Algorithm 1. Let $\Delta_I = \max_{i \in \mathcal{I}} |g^*(\{i\})|$ and $\Delta_T = \max_{t \in \mathcal{T}} |f^*(\{t\})|$. We define $\Delta = \max(\Delta_I, \Delta_T)$. Since the order of existences of integers k satisfying $I[k] \neq \emptyset$ is $O(\Delta^2)$, we can evaluate the number of **for** loops in our algorithm as $O(\Delta^2)$ by computing such k in advance. Because the computational complexity of each loop is $O(|I[k]|\Delta^2)$, the complexity of the entire computation can be estimated by $O(\sum_k |I[k]|\Delta^2)$. Because of the fact $\sum_k |I[k]| = O(\Delta^2)$, the complexity of the DFS function can be estimated as $O(\Delta^4)$. Consequently, to enumerate all new closed itemsets, the time complexity of our algorithm is $O(\Delta^4|NC|)$. Concerning the space complexity,

Algorithm 2 Updating sets of closed itemsets

```
1: function UPDATE($\mathcal{T}$, $\{t_1^*, \dots, t_n^*\}$)
2:     $\mathcal{T}^* \leftarrow \mathcal{T}$
3:     for $k = 1$ to $n$ do
4:         $\mathcal{T}^* \leftarrow (\mathcal{T}^* \cup \{t_k^*\})$
5:         ENUMERATE($\mathcal{T}^*, t_k^*$)
```

our algorithm do not hold any information about existing closed itemsets or those generated newly. This means that the space complexity of our algorithm can be regarded as *linear* with respect to the size of database.

4 Inserting Several Transactions Simultaneously

Next we take into account a situation in which we insert n transactions $t_1^*, t_2^*, \dots, t_n^* \subseteq \mathcal{I}^*$ simultaneously into our database $\mathcal{T}$. We let a new database inserted k out of n transactions be $\mathcal{T}_k^*$. It is obvious that if we already compute $C(\mathcal{T}_k^*)$, we can easily construct $C(\mathcal{T}_{k+1}^*)$ by applying our algorithm for inserting the transaction t_{k+1}^* into the database T_k^*. This means that we only need to apply our algorithm incrementally for inserting more than one transaction. We show such straightforward algorithm in Algorithm 2.

By using this simple algorithm, the time complexity of each update can be regarded as linear with respect to $|C(\mathcal{T}_k^*) \setminus C(\mathcal{T}_{k-1}^*)|$. Since it holds that $\sum_{k=1}^{n} |C(\mathcal{T}_k^*) \setminus C(\mathcal{T}_{k-1}^*)| = |C(\mathcal{T}_n^*) \setminus C(\mathcal{T}_0^*)|$, the time complexity of the entire updating processes can be also regarded as linear with respect to $|C(\mathcal{T}_n^*) \setminus C(\mathcal{T}_0^*)|$. Thus, we can still achieve our theoretical result: we can update sets of closed itemsets in linear with respect to the number of new closed itemsets which we need to enumerate.

5 Experimental Evaluations

We implemented our proposal and an existing method in C++, and evaluate the time required to update databases for several databases. Our experiments are done on a machine of Mac OS X 10.9 with a 12-core processor Xeon E5645 2.4 Ghz and 12GB main memory. Note that our source codes are compiled by using a compiler gcc 4.8.2. For experiments, we randomly generate databases consisting of N_T transactions with varying the number N_I of items, and the number N_O of items occurring in each transaction; that is, we set each transaction t has O items, which are selected by uniformly and randomly, from the entire set of items of cardinality N_I.

As a counterpart, we adopt an algorithm proposed by Godin [3], which is an algorithm to update concept lattices in FCA (Formal Concept Analysis). Since concepts studied in FCA are equivalent to closed itemsets in our setting, we can apply the Godin's algorithm directly. When we insert T transactions into a database $\mathcal{T}$, a database inserted first k transactions by $\mathcal{T}_k$ ($k = 0, 1, \dots, T$). Then, the time complexity of the algorithm for updating is $O(\Delta_I \sum_{k=0}^{T-1} |C(\mathcal{T}_k)|)$. The space complexity is given by $O(\Delta_I |C(\mathcal{T}_T)|)$ because their algorithm need to store every existing closed itemset.

In the following, we fix $T = 10000$. We vary $N_I = 128, 512, 2048, 8192, 16384, 32768$, and $O = 4, 8, 16, 32, 64$. For each parameter tuple (T, N_I, O), we randomly generate our database. From the empty database $\mathcal{T}_{\text{Init}} = \emptyset$, we update it by insering transactions from generated databases one by one. We compute times required to achieve generated random databases from the empty database $\mathcal{T}_{\text{Init}}$.

We show our results in Tables 5 and 6 and Figure 2. Since according to the growth of $|O|$ computational times also increase exponentially, we omit some entries in tables, where blank entries represent settings in which we halt algorithms by 1 hour. From these results, our algorithm is slower when $I = 128, 512$ but faster when $I = 2048, 8192, 16384, 32768$ comparing with the Godin's algorithm. That is why we can conjecture that our algorithm runs faster than the existing Godin's algorithm when O is large enough with respect to the number of transactions.

From the viewpoint of time complexity our algorithm depends on both Δ_I and Δ_T. On the other hand, the Godin's algorithm only depends on Δ_I. This fact also supports our experimental results because we randomly sample O items for each transaction in generating random databases.

6 Conclusion

In this paper we proposed a new algorithm to compute closed itemsets without computing it again when new transactions are inserted into our transaction databases. Our algorithm is designed by using

Table 5. Results by our proposal (msec)

O \ I	128	512	2048
4	20195	4273	984
8	450863	29914	3684
16	—	501767	33475
32	—	—	496527
O \ I	8192	16384	32768
4	516	453	419
8	960	720	648
16	3424	1708	1217
32	26451	7701	3663
64	408209	81275	21621

Table 6. Results by Godin's algorithm (msec).

O \ I	128	512	2048
4	5232	3563	2590
8	63338	21802	7568
16	2945891	209822	57934
32	—	—	643415
O \ I	8192	16384	32768
4	3107	3247	3188
8	6338	6962	7544
16	15508	14619	15527
32	104873	47374	37435
64	1485763	514126	194143

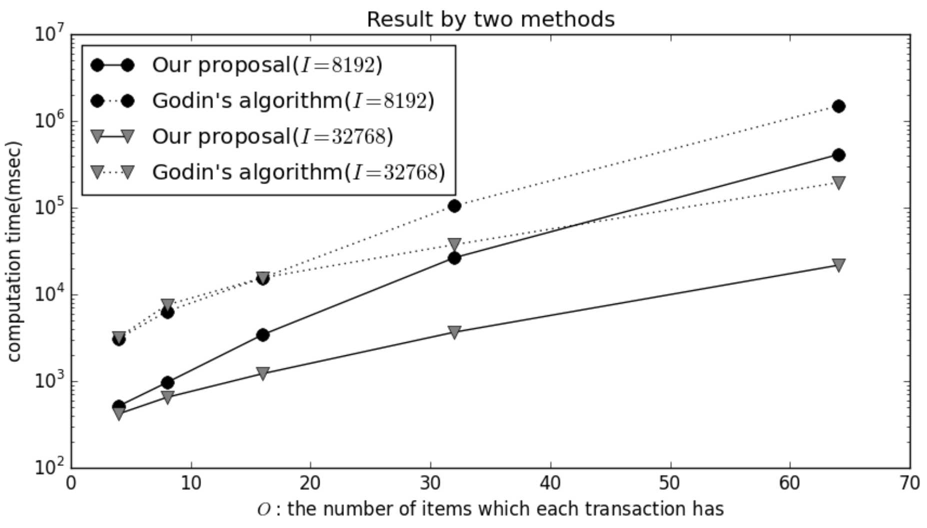

Fig. 2. Result by two methods

enumeration trees, and generates new closed itemsets by traversing such trees in a depth-first manner. The time complexity of our algorithm is linear with respect to the number of new closed itemsets. Precisely, for each new closed itemset, it requires $O(\Delta^4)$ time and linear space with respect to the size of transactions. We also shown experimental results by comparing our method with an well-known existing method proposed by Godin [3], which is originally designed in the area of FCA studies. The experimental results show that our algorithm is faster when I is large enough.

The proposed method enumerates not frequent closed itemsets but all frequent closed itemsets. Closed itemsets are equivalent to the intents of formal concepts in FCA(Formal Concept Analysis), so we can apply some methods proposed in FCA community also to enumerated closed itemsets. But when the purpose of enumeration is to obtain frequent itemsets, we reduce much computation time by enumerating only frequent closed itemsets. Our future work is to improve proposed method to design new updating algorithm to compute frequent closed itemsets.

Acknowledgement

This study was partially supported by JSPS KAKENHI Grant Number 22240010 and 26280085.

References

1. Chi Y., Wang, H., Yu P., and Richard M.: Moment: Maintaining closed frequent itemsets over a stream sliding window, *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, (2004), pp.59–66
2. Ganter B., and Wille, R.: *Formal Concept Analysis: Mathematical Foundations*, (1999)

3. Godin R. and Missaoui, R., and Hassan A.: Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices, *Computational intelligence*, vol. 11, issue 2 (1995), pp.246–267
4. Pasquier N., Bastide Y., Taouil R., and Lakhal L.: Efficient Mining of Association Rules Using Closed Itemset Lattices, *Information Systems*, vol. 24, issue 1, pp. 25–46 (1999)
5. Pei J., Han J., and Mao R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 21–30 (2000)
6. Uno T., Asai T., Uchida Y., and Arimura H.: LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets, *Proc. of Workshop on Frequent Itemset Mining Implementation*(*FIMI'03*), vol. 90 (2003)
7. Uno T., Kiyomi M., and Arimura H.: LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. *Proc. of Workshop on Frequent Itemset Mining Implementation*(*FIMI'04*), vol. 126 (2004)
8. Uno T., Kiyomi M., and Arimura H.: LCM ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. *Proc. of the 1st international workshop on open source data mining: frequent pattern mining implementations. ACM*, (2005)
9. Zaki M. J., and Hsiao C.: CHARM: An Efficient Algorithm for Closed Itemset Mining, *2nd SIAM International Conference on Data Mining*, pp. 457–473 (2002)