IPSJ Transactions on Databases Vol.5 No.3 15–25 (Sep. 2012)

Regular Paper

TripleEye: Mining Closed Itemsets with Minimum Length Thresholds Based on Ordered Inclusion Tree

Hiroyuki Shindo[1,a)] Tsutomu Hirao[1] Jun Suzuki[1] Akinori Fujino[1]
Masaaki Nagata[1]

Received: March 20, 2012, Accepted: July 2, 2012

Abstract: Itemset mining is one of the most essential tasks in the field of data mining. In this paper, we focus on minimum length as a mining measure for closed itemset mining. That is, our task is formalized as follows: Given a database and user-specified minimum length threshold, find all closed itemsets whose length is at least the minimum length. Closed itemset mining based on the minimum length threshold is preferable when it is difficult for users to determine the appropriate minimum support value. For our task, we propose TripleEye: an efficient algorithm of closed itemset mining that is based on the intersection of transactions in a database. Our algorithm utilizes the information of inclusion relations between itemsets to avoid the generation of duplicate itemsets and reduce the computational cost of intersection. During the mining procedure, the information of inclusion relations is maintained in a novel tree structure called Ordered Inclusion Tree. Experiments show that our algorithm dramatically reduces the computational cost, compared against naive intersection-based algorithm. Our algorithm also achieves up to twice the running speed of conventional algorithms given dense databases.

Keywords: itemset mining, minimum length, ordered inclusion tree

1. Introduction

Itemset mining is a fundamental task for knowledge discovery in databases. It has been widely applied to discover association rules from many kinds of databases such as product transactions recorded by point-of-scale (POS) system, Web click stream logs [4], and biological gene expressions [11], [13], [14]. The task of itemset mining is, in general, to find all interesting itemsets satisfying some constraint such as minimum support, minimum confidence [10], and minimum length threshold.

Most previous works [1], [9], [18], [21] have focused on frequent itemset mining, which uses a *minimum support* threshold to explore itemsets. In addition, frequent *closed* itemset mining has been proposed [15] as an alternative to standard frequent itemset mining to reduce the large number of frequent itemsets. An itemset x is a closed itemset if there exists no superset in the database whose frequency is the same as x. Closed itemsets contain complete information of all itemsets, but their size is usually much smaller [8]. The task of frequent closed itemset mining is formulated as follows: *Given a database and user-specified minimum support value, find all closed itemsets that satisfy the minimum support threshold*. As some researchers have pointed out [6], [17], however, the above framework has some drawbacks that hinder its practical use.

First, it is difficult to determine an appropriate minimum support value for a given database. Setting the value too small often makes the task computationally infeasible, while setting it too large may omit some interesting patterns. The FIMI workshop [4] on frequent itemset mining has shown that the number of frequent patterns in a database heavily depends on various factors, e.g., size, density, etc. Therefore, it takes a lot of time and effort for users to find a good threshold by trial and error [6].

Second, the output of frequent itemset mining contains a large number of excessively short patterns, e.g., itemsets containing only one item. This is because every subset of a frequent itemset is also a frequent itemset, which is well-known as the Apriori property [1]. Those too short patterns do not capture the relationship between items in which we are interested, and thus motivate us to find all itemsets under the condition of *minimum length* threshold, not minimum support.

In this paper, we focus on *minimum length* as a mining measure and formalize the alternative task of closed itemset mining as: *Given a database and user-specified minimum length value, find all closed itemsets whose length is at least the minimum length*. It should be noted that mining itemsets with minimum length constraint has been explored so far [2], however, their work differs from ours in that they present more general concept mining satisfying user-defined constraints.

Minimum length is a useful measure for closed itemset mining for several reasons. First, in most cases, there are relatively few minimum lengths (e.g., 1 to 30) [6], whereas minimum sup-

[1] NTT Communication Science Laboratories, Souraku, Kyoto 619–0237, Japan
[a)] shindo.hiroyuki@lab.ntt.co.jp

port values can range widely (e.g., 0.00% to 100.00%). Therefore, it is easy and intuitive to adjust the minimum length value for a database. Second, often the application offers some background knowledge about itemset length. For example, in biological databases, it is known that a transcription factor binding site has a length from 6 to 15 [20]. In such cases, we need an efficient algorithm for itemset mining that can satisfy a minimum length threshold without specifying minimum support. Third, closed itemsets whose length is at least minimum length contain sufficient information with regard to short sub-patterns. Therefore, the closed itemsets mined by our approach can be used for further data mining analysis without loss of information.

Our task is closely related to top-k frequent closed itemset mining [6], which is formulated as follows: *Given a database and user-specified k and minimum length, find the top-k frequent closed itemsets that satisfy the minimum length threshold*. In top-k frequent closed itemset mining, it is unnecessary to specify minimum support, however, setting the appropriate number of k is not as trivial as minimum support since it depends on the number of frequent itemsets in the database. In addition, top-k most frequent itemsets are usually not the most representative k patterns [8]. On the other hand, our task makes it unnecessary to specify k or minimum support.

Previous works on frequent closed itemset mining can be categorized as the enumeration-based approach [5], [16], [21] and the intersection-based approach [3], [12]. The former starts with a one-element itemset, then explores longer itemsets gradually in a depth-first or breadth-first manner. In contrast, the latter starts with large itemsets, then generates common itemsets by calculating the intersection of two or more transactions. Since our goal is to find closed itemsets whose length is at least minimum length, we expect the intersection-based approach to be the more efficient choice.

For our task, we propose *TripleEye*[*1]: an efficient closed itemset mining algorithm that satisfies the minimum length threshold. Our algorithm takes the intersection-based approach, but is much more efficient than the conventional alternatives [3], [12]. The key idea of our algorithm is that information of inclusion relations between itemsets is helpful in reducing computational cost. For example, the intersection of itemset {apple, orange} and superset {apple, banana, orange} is {apple, orange} itself, which is unnecessary for our task. Furthermore, if we observe that {apple, orange} and {cherry, peach, pear} have nothing in common, then {apple, orange} and the subset of {cherry, peach, pear}, e.g., {cherry, peach} also have nothing in common. Considering the examples stated above, we keep the information of inclusion relations in a novel tree structure called *Ordered Inclusion Tree*, and so avoid any unnecessary computational costs. Experiments show that our algorithm up to twice as fast as the conventional algorithms given dense databases.

Overall, our contributions are as follows:

(1) We propose Ordered Inclusion Tree: a compact tree structure for retaining the information of inclusion relations between itemsets.
(2) We propose TripleEye: an efficient algorithm for closed itemset mining, it satisfies the minimum length threshold by using Ordered Inclusion Tree.
(3) Experiments show that the running time of our algorithm is at least 50% less than the conventional algorithms given dense databases.

The rest of the paper is organized as follows: in Section 2, we discuss the related work. In Section 3, we introduce some preliminaries and give the formal task definition. In Section 4, we present a concept of Ordered Inclusion Tree and give the theory and algorithm of our method. In Section 5, we report the performance study of our algorithm, and in Section 6, we conclude our work.

2. Related Work

2.1 Enumeration-based Approach for Frequent Closed Itemset Mining

There has been much work on frequent enumeration-based closed itemset mining such as CLOSET [16], CHARM [21], CLOSET+ [19], and TFP [6], [7].

CLOSET is an enumeration-based algorithm for frequent closed itemset mining. It constructs a prefix tree structure called FP-tree, which represents compressed but complete information of frequent patterns in a database. CLOSET uses a depth-first search strategy and efficiently generates closed frequent itemsets with FP-tree.

CHARM is also an enumeration-based algorithm for frequent closed itemset mining. It maintains an original tree structure called IT-Tree (Itemset Tidset Tree), which contains both itemset and transaction id set simultaneously. CHARM also implements several optimization techniques such as *diffset*, a compact representation of IT-Tree, for memory efficiency and fast frequency computation.

CLOSET+ is another enumeration-based algorithm for frequent closed itemset mining. Similar to CLOSET, it makes use of FP-tree for efficient pattern generation and performs in a depth-first search. CLOSET+ integrates two types of tree-projection methods: bottom-up tree projection for dense databases and top-down tree projection for sparse databases. Owing to this hybrid tree-projection approach, it has been shown that CLOSET+ has advantages over CLOSET and CHARM in terms of runtime, memory and scalability.

FPclose is also an enumeration-based algorithm for frequent closed itemset mining with FP-tree structure. FPclose uses an additional simple array structure to traverse FP-trees efficiently, which greatly reduces the computational cost. In FIMI'03 workshop, FPclose won the best implementation award [4].

TFP [6] is a enumeration-based top-k frequent closed itemset mining algorithm. Similar to the other algorithms, TFP uses FP-tree structure for efficient itemset mining. TFP starts by setting the minimum support value to zero, then gradually raises minimum support with the search space pruning method called item merging and prefix itemset skipping. Experiments show that TFP outperforms CHARM and CLOSET+ if minimum support is well tuned.

[*1] TripleEye is derived from three "I"s: Itemset, Intersection and Inclusion.

2.2 Intersection-based Approach for Frequent Closed Itemset Mining

A simple intersection-based approach for frequent closed itemset mining has been proposed by Ref. [12], but it is too inefficient for real databases.

IsTa [3] is an alternative intersection-based approach for frequent closed itemset mining. It relies on a prefix tree structure for storing closed itemsets and quickly computing intersections. IsTa is much faster than the simple intersection-based implementation proposed by [12], and is also faster than the enumeration-based variants for some databases.

Our algorithm takes the intersection approach, but differs from the work of Ref. [12] and IsTa. Instead of relying on FP-tree or other prefix-tree structure variants, our algorithm constructs a novel data structure called Ordered Inclusion Tree to keep the information of inclusion relations between already mined closed itemsets.

3. Preliminaries

Let $F = \{f_1, f_2, \ldots, f_m\}$ be a set of discrete items. *Itemset* x is a subset of F, i.e., $x \subseteq F$. The *length* of itemset x, denoted by $|x|$, is the number of elements contained in x. We assume a database composed of a set of rows $R = \{r_1, r_2, \ldots, r_n\}$, where each row r_i is an itemset.

Example 3.1 *Table 1 shows an example database. In this example,* $F = \{a, b, c, d, e\}$ *and there are five rows* $R = \{r_1, \ldots, r_5\}$ *in the database. Row lengths are* $|r_1| = |r_2| = |r_3| = 4$ *and* $|r_4| = |r_5| = 3$.

Itemset x is referred to as a *closed itemset* in R if there exists no itemset x' such that $x \subset x'$ and $\forall i : (x \subseteq r_i \rightarrow x' \subseteq r_i)$. In other words, x is a closed itemset if there exists no superset in R which has the same frequency as x. In this work, we use an alternative and equivalent definition of closed itemset as follows [12]; it directly represents the relationship between closed itemsets and intersection of rows.

Definition 3.1 *Closed itemset*

Itemset x is a closed itemset in R if and only if there exists $I \subseteq \{1, 2, \ldots, n\}$ such that $x = \bigcap_{i \in I} r_i$, where n is the number of rows.

According to the definition 3.1, every row itself and all itemsets produced by the intersection of rows are closed itemsets.

Example 3.2 *In the example of Table 1, closed itemsets are* $\{a, b, c, d\} = r_1$, $\{b, c, d, e\} = r_2$, $\{a, b, d, e\} = r_3$, $\{b, c, d\} = r_4$, $\{a, d, e\} = r_5$, $\{a, b, d\} = r_1 \cap r_3$, $\{b, d, e\} = r_2 \cap r_3$, $\{a, d\} = r_1 \cap r_5$, $\{d, e\} = r_2 \cap r_5$, $\{b, d\} = r_3 \cap r_4$ *and* $\{d\} = r_4 \cap r_5$.

Task definition

Given a database consisting of R and user-specified minimum length threshold *minlen*, our task is to find all closed itemsets whose length is at least *minlen*.

A set of closed itemsets satisfying *minlen*, denoted by $C(R, minlen)$, can be represented as follows:

$$C(R, minlen) \equiv \{\textstyle\bigcap_{i \in I} r_i \,|\, I \wedge |\bigcap_{i \in I} r_i| \geq minlen\} \quad (1)$$

where $I \subseteq \{1, 2, \ldots, n\}$. According to Eq. (1), a straightforward implementation of intersection-based closed itemset mining that satisfies *minlen* is shown in algorithm 1, where $X \setminus Y$ (as in line 7) represents a relative complement of a set Y with respect to a set X.

Table 1 Example database.

i	r_i
1	a, b, c, d
2	b, c, d, e
3	a, b, d, e
4	b, c, d
5	a, d, e

Algorithm 1: Straightforward implementation of closed itemsets mining

Input : R, *minlen*
Output: a set of closed itemsets: C

```
   // initialize
1  R' ← {r ∈ R | |r| ≥ minlen}
2  A_0 ← R'
3  C_0 ← R'
4  Step k ← 0
5  while |A_k| > 0 do
6      Pick a ∈ A_k and remove a from A_k
       // calculate intersection
7      T ← {a ∩ c | c ∈ C_k ∧ |a ∩ c| ≥ minlen} \ C_k
8      A_{k+1} ← A_k ∪ T
9      C_{k+1} ← C_k ∪ T
10     k ← k + 1
11 end
12 C ← C_k
```

Algorithm 1 uses two sets of itemsets: A_k and C_k. Both A_k and C_k are first initialized as R' in line 2 and line 3. For each step, k, an arbitrary itemset $a \in A_k$ is picked, and then the intersection of a and $c \in C_k$ is calculated in line 7. All newly generated itemsets are added to A_k and C_k. Therefore, $C_k \supseteq A_k$ holds for any step k. When the number of elements in A_k becomes zero, the algorithm terminates. The output of algorithm 1 is equal to $C(R, minlen)$ in Eq. (1) (the proof is shown in Appendix A).

The algorithm 1 is extremely inefficient. This is because when a new closed itemset x is generated by intersection, the algorithm again calculates the intersection of x and every other itemset $c \in C_k$ (line 7). We propose a more efficient algorithm in the next section.

4. Proposed Method

In this section, we propose a novel data structure called Ordered Inclusion Tree, and then propose an efficient algorithm for our closed itemset mining task that uses Ordered Inclusion Tree.

4.1 Ordered Inclusion Tree

4.1.1 Theory and Algorithm

We introduce the concept of *Ordered Inclusion Tree*, which is a necessary data structure for our algorithm.

Definition 4.1 *Ordered Inclusion Tree*

Ordered Inclusion Tree (OI-Tree in short) is an ordered tree structure whose nodes are itemsets. The root node of OI-Tree is always set to F.

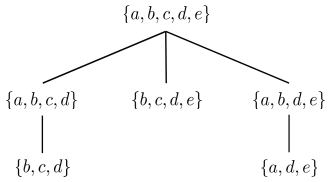


Fig. 1 Example OI-Tree constructed from the database in Table 1.

Let X be a set of itemsets. We denote OI-Tree as consisting of X as X^{OI}, that is, we handle X^{OI} as a set of itemsets with internal tree structure. X^{OI} satisfies the following conditions:

(1) $\forall x \in X^{OI} : x \subset Parent(x)$

(2) $\forall i, \forall j : \left(s_i \not\subseteq s_j \wedge s_i \not\supseteq s_j\right)$

(3) $\forall i, \forall j : \left(i < j \rightarrow \forall d \in Desc\left(s_j\right) : s_i \not\supseteq d\right)$

where $Parent(x)$ is a parent node of x, $Desc(x)$ is a set of descendant nodes of x. Likewise, $Children(x)$ is defined as a set of child nodes of x, and $Root\left(X^{OI}\right)$ is defined as the root node of X^{OI}. s_i and s_j $(i, j = 1, 2, \ldots)$ are arbitrary ordered sibling nodes in OI-Tree.

Example 4.1 ***Figure 1*** *shows an example of the OI-Tree constructed from the database in Table 1. The root node is $F = \{a, b, c, d, e\}$. $Parent(\{a, b, c, d\}) = \{a, b, c, d, e\}$, and $Desc(\{a, b, c, d\}) = \{\{b, c, d\}\}$. Furthermore, $\{a, b, c, d\} = s_1$, $\{b, c, d, e\} = s_2$, $\{a, b, d, e\} = s_3$ are example sibling nodes.*

$\{b, c, d\}$ is a proper subset of both $\{a, b, c, d\}$ and $\{b, c, d, e\}$. According to the third condition of OI-Tree, $\{b, c, d\}$ must be a child node of the first superset $\{a, b, c, d\}$.

OI-Tree clearly represents inclusion relations between itemsets. The first condition of OI-Tree requires the parent node of itemset x to be a proper superset of x. The second condition prevents sibling nodes in OI-Tree from containing each other. The third condition requires itemset x to be the descendant node of the first superset in sibling nodes. For example, assume $x \subset s_i\,(i = 1, 2, \ldots)$ where $\{s_i\}$ are sibling nodes in OI-Tree. To satisfy the third condition, x must be the descendant node of s_1. Due to the third condition, OI-Tree is well-organized even when x has multiple supersets in X.

X^{OI} can be efficiently constructed in a top-down manner by adding an itemset to X^{OI} in length descending order. This is because the length of the parent node is always longer than that of a child node. We show the basic procedure of constructing X^{OI} in algorithm 2. Note that given a set of itemsets X, X^{OI} is not uniquely determined. It depends on in what order an itemset is added to X^{OI}.

The algorithm 2 for constructing OI-Tree calls the procedure *FindParent*, which takes two arguments and returns the parent node of the second argument. The procedure*FindParent* is shown in the algorithm 3.

4.1.2 Terms

We define some terms related to OI-Tree: *Pos*, *FirstPos*, and *RightSibDesc* to permit further discussion.

Definition 4.2 *Pos*

Let X^{OI} be an OI-Tree. For itemset $x \in X^{OI}$, $Pos(x)$ (abbreviation of Position) is a numerical value numbered in ascending order according to the depth-first traversal of X^{OI}. The Pos value of the root node is set to 0*.*

Algorithm 2: ConstructOITree

Input : a set of itemsets: X
Output: X^{OI}

```
// initialize
Root(X^OI) ← F
while |X| > 0 do
    x ← argmax_{x'∈X} |x'| and remove x from X
    p ← call FindParent(Root(X^OI), x)
    Add x to the last child node of p
end
```

Algorithm 3: FindParent

Input : candidate parent node: p, target node: x $(p \supset x)$
Output: $Parent(x)$

```
foreach c ∈ Children(p) do
    if c ⊃ x then
        return call FindParent(c, x)
    end
end
return p
```

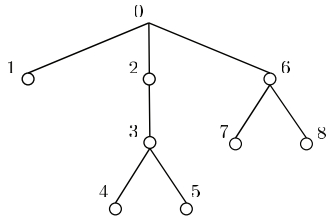


Fig. 2 The *Pos* value for each itemset in an OI-Tree.

Example 4.2 ***Figure 2*** *shows $Pos(x)$ value in an OI-Tree. Each itemset in the nodes are omitted for simplicity. The $Pos(x)$ value for each itemset x is shown by the index at the upper of the node.*

Definition 4.3 $FirstPos$

Let X^{OI} be an OI-Tree and let Y be a set of itemsets that satisfies $Y \subseteq X^{OI}$. Then, $FirstPos(Y)$ (abbreviation of First Position) is defined as follows:

$$FirstPos(Y) \equiv \underset{y \in Y}{argmin}\ Pos(y)$$

That is, $FirstPos(Y)$ is the itemset $y \in Y$ that minimizes its Pos value.

Definition 4.4 $RightSibDesc$

Let X^{OI} be an OI-Tree. For itemset $x \in X^{OI}$, $RightSibDesc(x)$ (abbreviation of Right Sibling Descendants) is a set of itemsets defined as follows:

$$RightSibDesc(x) \equiv \{y \,|\, Pos(y) > Pos(x) \wedge y \in Desc(Parent(x)) \wedge y \notin Desc(x)\}$$

Example 4.3 ***Figure 3*** *shows $RightSibDesc(x)$ in an OI-Tree. Itemsets in the nodes are omitted for simplicity. In Fig. 3, $RightSibDesc(x)$ is the circled set of three itemsets.*

4.2 TripleEye

We propose *TripleEye*: an efficient algorithm of closed itemset mining that satisfies *minlen* and uses OI-Tree. The straightforward implementation of Eq. (1) (shown in algorithm 1) is ex-

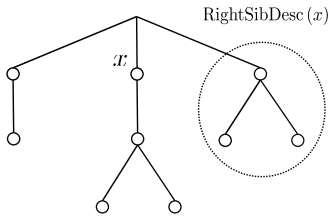


Fig. 3 *RightSibDesc* (x).

Algorithm 4: Closed itemset mining with OI-Tree

Input : R, *minlen*
Output: a set of itemsets: C

```
   // initialize
1  R' ← {r ∈ R | |r| ≥ minlen}
2  A_0 ← R'
3  C_0^OI ← call ConstructOITree(R')
4  Step k ← 0
5  while |A_k| > 0 do
6      a ← FirstPos(A_k) and remove a from A_k
       // calculate intersection
       T ← {a ∩ d_a | d_a ∈ RightSibDesc(a)
7          ∧ |a ∩ d_a| ≥ minlen} \ C_k^OI
8      A_{k+1} ← A_k ∪ T
       // grow OI-Tree
9      Sort T into length descending order
10     foreach t ∈ T do
11         p ← call FindParent(Root(C_k^OI), t)
12         Add t to the last child node of p
13     end
14     C_{k+1}^OI ← C_k^OI
15     k ← k + 1
16 end
17 C ← all nodes of C_k^OI
```

tremely inefficient. This is because when a new closed itemset x is generated by intersection, the algorithm again calculates the intersection of x and every other itemset $c \in C_k$ (line 7 in algorithm 1).

The main idea of our algorithm is as follows: when a new closed itemset x is generated by intersection, x is added to the OI-Tree composed of C_k. This makes it sufficient to calculate only the intersection of x and $d_x \in RightSibDesc(x)$, which is guaranteed by the *Localization* theorem (described later). Since the number of itemsets in $RightSibDesc(x)$ is smaller than that of itemsets in C_k, the computational cost is reduced. We also show that the intersection of x and $d_x \in RightSibDesc(x)$ can be efficiently calculated by using the property of OI-Tree.

4.2.1 Algorithm

Consider an OI-Tree for C_k, i.e., C_k^{OI} in algorithm 1. We modify algorithm 1 as shown in algorithm 4.

Algorithm 4 is different from algorithm 1 in three ways. First, C_k in algorithm 1 yields OI-Tree C_k^{OI}. Second, in line 6, $a = FirstPos(A_k)$ is picked for each step, whereas an arbitrary itemset is picked in algorithm 1. Third, in line 7, the intersection of a and only $d_a \in RightSibDesc(a)$ is calculated, whereas the intersection of a and every other itemset in C_k is calculated in algorithm 1. For each step, the *Pos* value of each itemset in OI-Tree is updated with each additional node in C_k^{OI}.

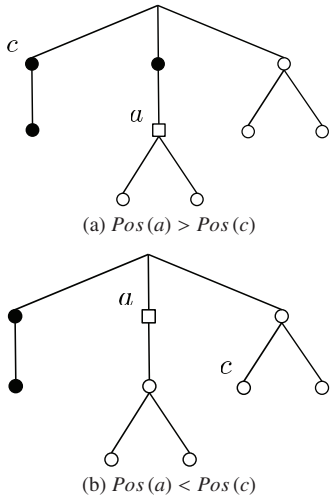


Fig. 4 The examples of OI-Tree which represent (a) $Pos(a) > Pos(c)$ and (b) $Pos(a) < Pos(c)$.

4.2.2 Localization Theorem

We show two lemmas and then give the Localization theorem to validate line 7 of algorithm 4.

Lemma 4.1 *In algorithm 4, let $a \in A_k$, $c \in C_k^{OI}$ be closed itemsets for arbitrary step k. For any step k, $A_k \subseteq C_k^{OI}$ is satisfied. In line 6, $a = FirstPos(A_k)$ is picked, then*

$$\forall k : \left(Pos(a) > Pos(c) \wedge |a \cap c| \geq minlen \rightarrow a \cap c \in C_k^{OI}\right)$$

Figure 4 (a) shows the example of C_k^{OI} which represents $Pos(a) > Pos(c)$. The itemsets in white nodes are elements of A_k, and the itemsets in black nodes are elements of $C_k \setminus A_k$. The itemset in the square node represents $a = FirstPos(A_k)$.

Proof.

If $a \in Desc(c)$, $a \cap c = a$. Hence the given lemma holds.

Let us assume $a \notin Desc(c)$. A closed itemset can be represented as the intersection of rows, that is, $\exists I \subseteq \{1, 2, \ldots, n\}$: $a \cap c = (\cap_{i \in I} r_i) \cap c$.

Considering the assumption $Pos(c) < Pos(a)$, for step k' $(0 < k' < k)$, the intersection of c and every other itemset in $C_{k'}^{OI}$ is calculated in line 6. Since $\forall r \in R' : r \in C_0^{OI}$,

$$\forall i : \left(|r_i \cap c| \geq minlen \rightarrow r_i \cap c \in C_k^{OI}\right)$$

Moreover, $Pos(r_i \cap c) < Pos(a)$ is satisfied since $r_i \cap c \subseteq c$. Consequently,

$$\forall i, \forall j : \left(\left|r_i \cap r_j \cap c\right| \geq minlen \rightarrow r_i \cap r_j \cap c \in C_k^{OI}\right)$$

Likewise, if $|a \cap c| \geq minlen$, then $a \cap c = (\cap_{i \in I} r_i) \cap c \in C_k^{OI}$. □

Lemma 4.2 *In algorithm 4, let $a \in A_k$, $c \in C_k^{OI}$ be closed itemsets for arbitrary step k. For any step k, $A_k \subseteq C_k^{OI}$ is satisfied. In line 6, $a = FirstPos(A_k)$, then*

$$\forall k : \left(Pos(a) < Pos(c) \wedge |a \cap c| \geq minlen \wedge a \cap c \notin C_k^{OI} \rightarrow a \cap c \in \{a \cap d_a \mid d_a \in RightSibDesc(a)\}\right)$$

Figure 4 (b) shows the example of C_k^{OI} which represents $Pos(a) < Pos(c)$. The itemsets in white nodes are elements of A_k, and the itemsets in black nodes are elements of $C_k \setminus A_k$. The itemset in the square node represents $a = FirstPos(A_k)$.

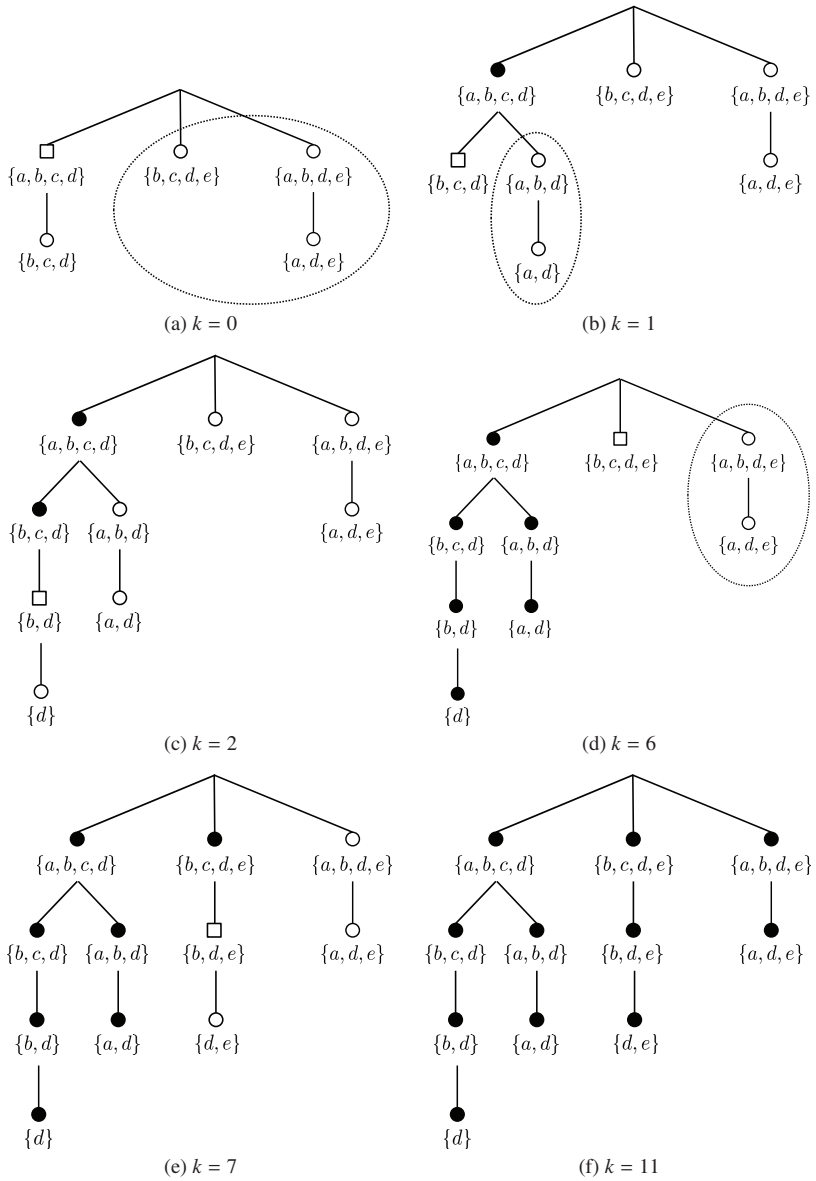


Fig. 5 Step-by-step example of our algorithm. The itemsets in white nodes are elements of A_k, and the itemsets in black nodes are elements of $C_k \setminus A_k$. The itemset in the square node represents $a = FirstPos(A_k)$. $RightSibDesc(a)$ is a set of circled itemsets.

Proof.

If $c \in Desc(a)$ then $a \cap c = a$, which contradicts the supposition $a \cap c \notin C_k^{OI}$.

Let us assume $c \notin Desc(a)$. $a \cap c$ can be rewritten as follows:

$$a \cap c = (a \cap Parent(a)) \cap c = a \cap (Parent(a) \cap c)$$

Let $c' \equiv Parent(a) \cap c$. Here we prove $\forall k : c' \in C_k^{OI}$ and $Pos(a) < Pos(c')$ is satisfied.

When the given assumptions $a \cap c \notin C_k^{OI}$ and $Pos(a) < Pos(c)$ are satisfied,

$$\forall d_a \in RightSibDesc(a) : Pos(a \cap c) < Pos(d_a)$$

holds since $a \cap c \subseteq a$. Hence, c is not the intersection of any two itemsets, i.e., $\exists r_i \in R' : c = r_i$. Therefore, $\forall k : c' \in C_k^{OI}$ holds.

According to lemma 4.1, if $Pos(a) > Pos(c')$ and $|a \cap c'| \geq minlen$, then $a \cap c' \in C_k^{OI}$, which contradicts the supposition $a \cap c' = a \cap c \notin C_k^{OI}$. Therefore, $Pos(a) < Pos(c')$.

Consequently, c' satisfies $c' \subseteq Parent(a)$, and $Pos(a) < Pos(c')$, thus

$$c' = a \vee c' \in Desc(a) \vee c' \in RightSibDesc(a)$$

If $c' = a \vee c' \in Desc(a)$ then $a \cap c' = a$, which contradicts the supposition $a \cap c' = a \cap c \notin C_k^{OI}$. Hence, $a \cap c' \in \{a \cap d_a \,|\, d_a \in RightSibDesc(a)\}$ holds. □

Theorem 4.1 *Localization*

In line 7 of algorithm 4, let $a \in A_k$, $c \in C_k^{OI}$ be closed itemsets for arbitrary step k, then

$$\forall k : \left\{a \cap c \,\middle|\, c \in C_k^{OI} \wedge |a \cap c| \geq minlen\right\} \setminus C_k^{OI}$$
$$= \{a \cap c \,|\, c \in RightSibDesc(a) \wedge |a \cap c| \geq minlen\} \setminus C_k^{OI}$$

Proof.

If $a \cap c \in C_k^{OI}$ then $(a \cap c) \setminus C_k^{OI} = \emptyset$, thus the given theorem is true.

Let us assume $a \cap c \notin C_k^{OI}$. According to lemma 4.1 and lemma 4.2, if $|a \cap c| \geq minlen$ then $Pos(a) < Pos(c)$ and

$a \cap c \in \{a \cap d_a | d_a \in RightSibDesc(a)\}$. Therefore, the given theorem is true. □

According to the localization theorem, the output of algorithm 4 is equal to that of algorithm 1. Since the number of itemsets in $RightSibDesc(a)$ is smaller than that of the itemsets in C_k^{OI}, the computational cost of intersection is reduced.

4.2.3 Step-by-Step Example

We give a step-by-step example of our algorithm. Let us use the database in Table 1 and set $minlen = 1$. **Figure 5** shows the transitions in C_k^{OI} yielded by running algorithm 4. Each subfigure represents C_k^{OI} for some step k.

For step $k = 0$, C_0^{OI} is constructed from R' (Fig. 5 (a)). In line 7 of algorithm 4, $a = FirstPos(A_0) = \{a, b, c, d\}$ is picked. Then, in line 7, the intersection of a and $d_a \in RightSibDesc(a)$ is calculated, that is, $\{a, b, c, d\} \cap \{b, c, d, e\}$, $\{a, b, c, d\} \cap \{a, b, d, e\}$, and $\{a, b, c, d\} \cap \{a, d, e\}$. Finally, all new itemsets generated by intersection are added to C_0^{OI} (see Fig. 5 (b)).

Similarly, for step $k = 1$, $a = FirstPos(A_1) = \{b, c, d\}$. Next, the intersections $\{b, c, d\} \cap \{a, b, d\}$ and $\{b, c, d\} \cap \{a, d\}$ are calculated, and added to C_1^{OI}. When $|A_k| = 0$, the algorithm terminates (Fig. 5 (f)). Then, the itemsets contained in C_k^{OI} are all of the closed itemsets that satisfy *minlen*.

4.3 Further Speedup

We introduce further speedup techniques for our algorithm.

4.3.1 Empty Itemsets by Intersection

The efficiency of our algorithm relies on how rapidly the intersection $a \bigcap d_a$ can be calculated (line 7 of algorithm 4), where $a \in A_k$ and $d_a \in RightSibDesc(a)$. The naive implementation of algorithm 4 generates empty itemsets many times. This is because if the intersection of itemsets x and y is an empty set, the intersection of x and the subset of y is also an empty set. To eliminate such unnecessary computational cost, we utilize the OI-Tree property as shown in the following lemma.

Lemma 4.3 *Let C_k^{OI} be an OI-Tree for step k in algorithm 4. If $a \in A_k$, $c \in C_k^{OI}$ are itemsets, then*

$$\forall d_c \in Desc(c) : \left(a \bigcap c = \emptyset \rightarrow a \bigcap d_c = \emptyset\right)$$

Proof.

According to the definition of OI-Tree, $\forall d_c \in Desc(c) : c \supset d_c$. Hence $a \bigcap c = \emptyset \rightarrow a \bigcap d_c = \emptyset$ holds. □

Owing to lemma 4.3, we can stop the depth-first search of OI-Tree in line 7 if $a \cap c = \emptyset$, which reduces the cost of intersection computation.

4.3.2 Binarized Database

The implementation of our algorithm requires the following three procedures:

- Share: checks whether $x \bigcap y \neq \emptyset$.
- Contain: checks whether $x \supseteq y$.
- Intersect: returns $x \bigcap y$.

where x and y are itemsets.

To execute those procedures quickly, we transpose the database into binary representation. **Table 2** shows the example of a binarized form of the database in Table 1. A binarized database allows us to compute the above procedures by bitwise operations as shown in the following lemma.

Table 2 Binarized database.

	a	b	c	d	e
r_1	1	1	1	1	0
r_2	0	1	1	1	1
r_3	1	1	0	1	1
r_4	0	1	1	1	0
r_5	1	0	0	1	1

Lemma 4.4 *If x and y are binary represented itemsets, then*

- *Share:* $x \bigcap y \neq \emptyset \Leftrightarrow xANDy \neq \emptyset$
- *Contain:* $x \supseteq y \Leftrightarrow xXOR(xORy) = \emptyset$
- *Intersect:* $x \bigcap y = xANDy$

where AND, XOR, and OR are bitwise operators of conjunction, exclusive disjunction, and disjunction, respectively.

Proof.

Obviously, *Share* and *Intersect* are true. The truth table of *Contain* is shown below:

x_k	y_k	x_k XOR (x_k OR y_k)
0	0	0
0	1	1
1	0	0
1	1	0

where x_k and y_k are the kth bits of x and y, respectively. According to the truth table, only if $x_k = 0$ and $y_k = 1$, does the operator yield 1. Therefore, *Contain* is true. □

5. Experiments

In our first experiment, we compare the computational cost of TripleEye against that of the algorithm 1 to evaluate the efficiency of our algorithm. Both algorithms are intersection-based approaches for closed itemset mining satisfying *minlen* threshold. However, TripleEye uses the OI-Tree structure for reducing the number of intersection operations, while the algorithm 1 has no such a structure.

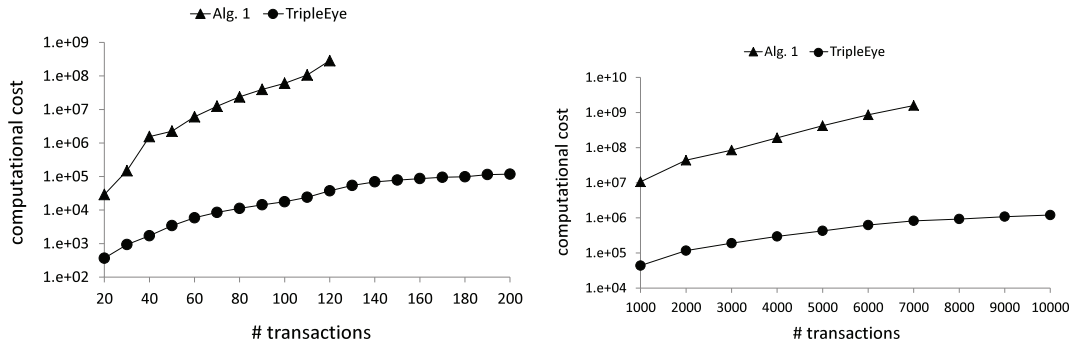
In our second experiment, we measure the running time of TripleEye with varying *minlen* threshold to evaluate whether TripleEye solves our task in practical time. Since most previous works are designed for closed itemset mining satisfying not *minlen* but minimum support, it is difficult to directly compare TripleEye agianst other algorithms. However, these conventional algorithms can be used for solving our task, thus we also show the running time of FPclose [5] and IsTa [3] as references.

FPclose is an enumeration-based algorithm of frequent closed itemset mining, and won the FIMI'03 best implementation award [4]. We downloaded the source code of FPclose from FIMI'03 repository[*2], and compiled it using Microsoft Visual C++ 2010. IsTa is a recently proposed intersection-based algorithm of frequent closed itemset mining. We downloaded the windows binary executable of IsTa from the author's website[*3].

In our third experiment, we measure the running time of TripleEye with varying the number of transactions to evaluate the scalability of our algorithm.

All experiments were conducted on a Core i7 3.2 Ghz CPU, running Windows® 7. We confirmed all implementations were

[*2] http://fimi.ua.ac.be/fimi03/
[*3] http://www.borgelt.net/ista.html



(a) chess (b) gazelle

Fig. 6 Computational cost comparison.

Table 3 Database characteristics.

Database	# Trans.	# Items	Avg. Length	Dense/Sparse
chess1k	1000	75	37	Dense
chess1.2k	1200	75	37	Dense
L15I40T3k	3000	40	15	Dense
L15I40T5k	5000	40	15	Dense
gazelle	59601	497	2.5	Sparse
T10I4D50K	50000	869	10.1	Sparse

compiled as windows 32 bit executables, thus the maximum amount of memory is limited to 3.12 GB.

We prepared both real and synthetic databases for the performance comparison. **Table 3** shows the characteristics of each database. Each database can be categorized as dense or sparse.

As a pre-experiment, we applied our speedup techniques (early stopping of depth-first search and database binarization in Section 4.3) to our algorithm and confirmed significant speedup. Therefore, those techniques are employed in all of our experiments.

5.1 Comparison against Simple Intersection-based Algorithm

Figure 6 shows the computational cost comparison of TripleEye (algorithm 4) against the simple intersection-based approach (algorithm 1) on the chess1k database (Fig. 6 (a)) and the gazelle database (Fig. 6 (b)). Both figures are plotted with computational cost as the vertical axis (logarithmic scale) and the number of transactions as the horizontal axis. The computational cost was measured by the number of intersection operations which were needed for finding all closed itemset mining satisfying *minlen* = 1.

It is apparent from Fig. 6 that TripleEye dramatically reduces the number of intersection operations. For example, the number of intersection operations of TripleEye was 37447, while that of algorithm 1 was 286037651 for 120 transactions on Fig. 6 (a). When the number of transactions exceeded 120, the algorithm 1 failed to calculate all closed itemsets in practical time. On the other hand, TripleEye successfully calculated all closed itemsets on more than 120 transactions because of the efficient mining procedure by using OI-Tree.

5.2 Running Time Comparison

5.2.1 Dense Databases

We used chess and L15I40 databases to compare the running time of mining algorithms for dense databases. The chess database was obtained from the UCI Machine Learning Repository[*4]. We selected the first 1k and 1.2k transactions (chess1k and chess1.2k). L15I40 database was created by the IBM data set generator. We set the number of transactions to 3k and 5k (L15I40T3k and L15I40T5k).

Figure 7 (a) shows the running time of TripleEye, IsTa, and FPclose on the chess1k database for *minlen* values from 1 to 35. The minimum support of IsTa and FPclose was set to 1. Therefore, when *minlen* is set to be 1, all algorithms produce the same output. We can see that TripleEye consistently outperformed IsTa and FPclose for any *minlen*.

Since IsTa and FPclose aim to find closed itemsets that satisfy minimum support, the running time of those algorithms is constant when *minlen* is ranged. Indeed, IsTa has an option of setting *minlen*, however, the running time was constant on the chess1k database regardless of *minlen*. On the other hand, the running time of TripleEye successfully decreased as *minlen* became longer.

The chess1k and chess1.2k database had the following properties: a) for minlen values from 1 to 15, the number of closed itemsets was almost constant. b) for minlen values from 15 to 35, the number of closed itemsets gradually decreased as *minlen* became longer. Therefore, we can see that TripleEye performs in accordance with the number of closed itemsets satisfying *minlen*.

Figure 7 (b) shows the running time of TripleEye and IsTa on the chess1.2k database for *minlen* values from 1 to 35. FPclose failed to mine closed itemsets because of a memory overflow error. As in Fig. 7 (a), our algorithm consistently outperformed IsTa regardless of *minlen*. Even when *minlen* was set to 1, our algorithm was approximately 1.5 times faster than IsTa.

Figures 7 (c) and 7 (d) show the running time of the three algorithms for *minlen* values from 1 to 15. As in the experiments on chess databases, our algorithm outperformed the conventional algorithms regardless of *minlen*. Even when *minlen* was 1, our algorithm was more than twice as fast as IsTa on the L15I405k database. Overall, our algorithm outperformed the other algorithms on dense databases, and efficiently found all closed itemsets satisfying *minlen.*

5.2.2 Sparse Databases

We used gazelle and T10I4D50K as sparse databases. Gazelle

[*4] http://archive.ics.uci.edu/ml/

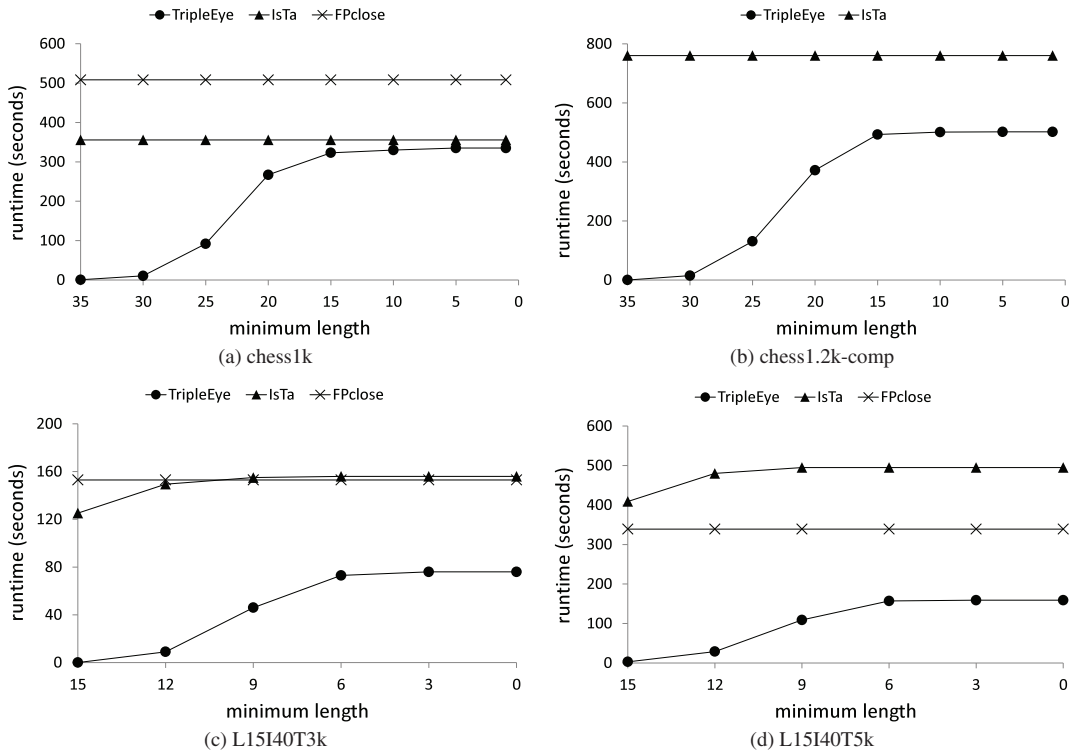


Fig. 7 Running time comparison on dense databases.

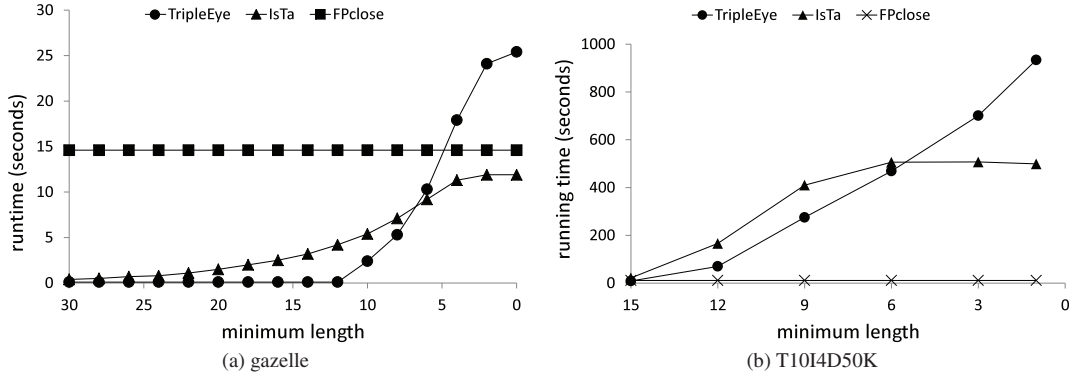


Fig. 8 Running time comparison on sparse databases.

holds click stream data [6], and T10I4D50K was created by the IBM data set generator.

Figure 8 (a) shows the running time of three algorithms on gazelle database. When *minlen* was relatively large (5 to 30), our algorithm was the fastest of the algorithms. When *minlen* was less than 5, our algorithm matched the results of IsTa and FPclose.

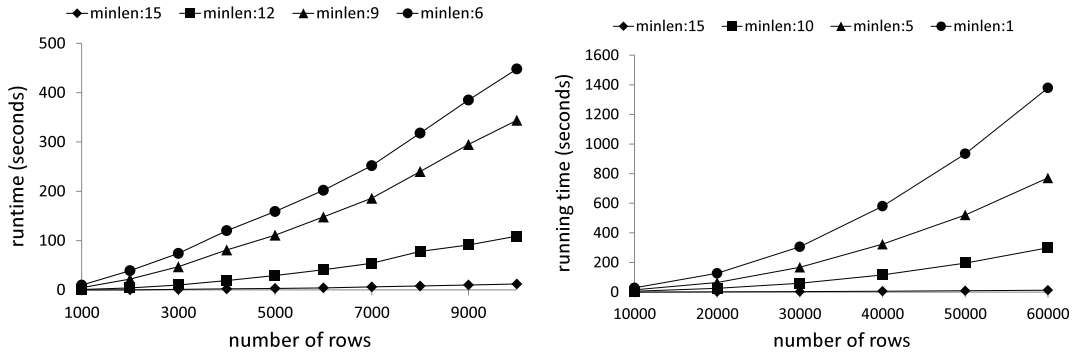
Figure 8 (b) shows the running time of the three algorithms on the T10I4D50K database. For the T10I4D50K database, FPclose was the fastest of the three algorithms. The average length of the T10I4D50K database is longer than that of the gazelle database. Furthermore, the length variance of the T10I4D50K database was considerably lower than that of the gazelle database. Therefore, it seems possible that the performance gap of intersection-based approaches (TripleEye and IsTa) between Fig. 8 (a) and Fig. 8 (b) is due to the average length and length variance, while the running time of enumeration-based approach (FPclose) is insensitive for those factors on sparse databases. TripleEye was faster than IsTa when *minlen* was relatively large, but slower than IsTa when *minlen* was small.

These results on dense and sparse databases can be explained in part by the difference between intersection-based and enumeration-based approach. An intersection-based approach generates closed itemsets in *decreasing* order of length. Therefore, our algorithm can prune all closed itemsets whose length are less than *minlen*, which resulted in better performance than FPclose and IsTa. IsTa is also intersection-based approach, however, the data structure and algorithm is designed for fast calculation of closed itemsets satisfying minimum support threshold.

On the other hand, an enumeration-based approach generates closed itemsets in *ascending* order of length. This is equivalent to itemsets generation in decreasing order of support value. Therefore, enumeration-based approach is suitable for pruning itemsets whose support value are less than minimum support, rather than *minlen*.

5.3 Scalability Test

We tested the scalability of our algorithm on dense and sparse



(a) TL15I40 (b) T10I4D50K

Fig. 9 Scalability test.

databases. **Figure 9** (a) shows the running time of TripleEye on the L15I40 database with four different *minlen* values. The number of transactions ranges from 1k to 10k. We can see that the running time of TripleEye increases almost linearly on the L15I40 database in all cases.

Figure 9 (b) shows the running time of TripleEye on the T10I4D50K database with four different *minlen* values. The number of transactions in the figure ranges from 10k to 60k. The figure shows that the running time of TripleEye increases almost linearly on the T10I4D50K database in four cases. Overall, it has been shown that our algorithm is scalable on dense and sparse databases in terms of database size.

6. Conclusions

We have studied the task of closed itemset mining where the constraint is at least matching the minimum length. Different from frequent closed itemset mining and top-k frequent closed itemset mining, our task formulation makes it unnecessary to specify minimum support or the value of k. Furthermore, it is possible to apply background knowledge about the length of itemsets to be mined. For example, in biological databases, it is known that a transcription factor binding site has a length from 6 to 15 [20]. In this case, we simply set *minlen* value to 6 without specifying minimum support. To reach our goal, we proposed the Ordered Inclusion Tree; it maintains the inclusion relations between itemsets, and also proposed an efficient algorithm, TripleEye, for closed itemset mining that can satisfy the minimum length threshold. Our algorithm utilizes the Ordered Inclusion Tree for reducing the computation cost of intersection operations. The validity of which is guaranteed by the Localization theorem. Experiments showed that our algorithm was up to twice as fast as the conventional alternatives given dense databases and minimum length of 1. We will extend our algorithm to support sequence, tree, and graph pattern mining.

References

[1] Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules, *Proc. 20th International Conference on Very Large Data Bases* (*VLDB*), Vol.1215, pp.487–499 (1994).

[2] Besson, J., Robardet, C., Boulicaut, J. and Rome, S.: Constraint-based concept mining and its application to microarray data analysis, *Intelligent Data Analysis*, Vol.9, No.1, pp.59–82 (2005).

[3] Borgelt, C., Yang, X., Nogales-Cadenas, R., Carmona-Saez, P. and Pascual-Montano, A.: Finding closed frequent item sets by intersecting transactions, *Proc. 14th International Conference on Extending Database Technology* (*EDBT/ICDT*), pp.367–376 (2011).

[4] Goethals, B. and Zaki, M.J.: FIMI '03: Workshop on frequent itemset mining implementations, *Proc. ICDM Workshop on Frequent Item Set Mining Implementations*, pp.1–13 (2003).

[5] Grahne, G. and Zhu, J.: Efficiently using prefix-trees in mining frequent itemsets, *Proc. ICDM Workshop on Frequent Itemset Mining Implementations* (2003).

[6] Han, J., Lu, Y. and Tzvetkov, P.: TFP: An efficient algorithm for mining top-k frequent closed itemsets, *IEEE Trans. Knowledge and Data Engineering*, Vol.17, No.5, pp.652–664 (2005).

[7] Han, J., Wang, J., Lu, Y. and Tzvetkov, P.: Mining top-k frequent closed patterns without minimum support, *Proc. IEEE International Conference on Data Mining* (*ICDM*), pp.211–218 (2002).

[8] Han, J., Cheng, H., Xin, D. and Yan, X.: Frequent pattern mining: current status and future directions, *Data Mining and Knowledge Discovery*, Vol.15, No.1, pp.55–86 (2007).

[9] Han, J., Pei, J. and Yin, Y.: Mining frequent patterns without candidate generation, *Proc. ACM SIGMOD International Conference on Management of Data*, pp.1–12 (2000).

[10] Kim, W.-Y., Lee, Y.-K. and Han, J.: CCMine: Efficient mining of confidence-closed correlated patterns, *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD*), pp.569–579 (2004).

[11] Liu, H., Han, J., Xin, D. and Shao, Z.: Top-down mining of interesting patterns from very high dimensional data, *Proc. International Conference on Data Engineering* (*ICDE*) (2006).

[12] Mielikäinen, T.: Intersecting data to closed sets with constraints, *Proc. ICDM Workshop on Frequent Item Set Mining Implementations* (2003).

[13] Pan, F., Cong, G., Tung, A.K.H., Yang, J. and Zaki, M.J.: Carpenter: Finding closed patterns in long biological datasets, *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD*), pp.637–642 (2003).

[14] Pan, F., Tung, A.K.H., Cong, G. and Xu, X.: COBBLER: Combining column and row enumeration for closed pattern discovery, *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD*) (2004).

[15] Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L.: Discovering frequent closed itemsets for association rules, *Proc. International Conference on Database Theory* (*ICDT*), Vol.1540, pp.398–416 (1999).

[16] 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*, Vol.4, No.2, pp.21–30 (2000).

[17] Sese, J. and Morishita, S.: Answering the most correlated N association rules efficiently, *Proc. 6th European Conference on Principles and Practice of Knowledge Discovery in Databases* (*PKDD*), Lecture Notes in Artificial Intelligence, Vol.2431, pp.410–422 (2002).

[18] Uno, T., Asai, T., Uchida, Y. and Arimura, H.: LCM: An efficient algorithm for enumerating frequent closed item sets, *Proc. ICDM Workshop on Frequent Item Set Mining Implementations* (2003).

[19] Wang, J., Han, J. and Pei, J.: CLOSET+: Searching for the best strategies for mining frequent closed itemsets, *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD*), pp.236–245 (2003).

[20] Wang, K., Xu, Y. and Yu, J.X.: Scalable sequential pattern mining for biological sequences, *Proc. 13th ACM Conference on Information and Knowledge Management* (*CIKM*), pp.178–187 (2004).

[21] Zaki, M.J. and Hsiao, C.J.: CHARM: An efficient algorithm for closed itemset mining, *Proc. SIAM International Conference on Data Mining* (*SDM*), Vol.15 (2002).

Appendix

A.1 Correctness of the Algorithm 1

Here, we prove that the output of algorithm 1 is equal to $C(R, minlen)$ in Eq. (1). Equation (1) is a definition of closed itemset mining satisfying *minlen*. The algorithm 1 is a simple intersection-based approach for calculation of Eq. (1). We employ the following two lemmas for our proof.

Lemma A.1.1 *Let C be the output of algorithm 1, then*

$$\forall c \in C : c \in C(R, minlen)$$

Proof.

Basis: In algorithm 1, $C_0 = R'$. Hence $\forall c \in C_0 : c \in C(R, minlen)$.

Induction: Assume that for arbitrary step k, $\forall c \in C_k : c \in C(R, minlen)$ in line 6. By using the induction hypothesis, $\forall a \in A_k : a \in C(R, minlen)$ since $A_k \subseteq C_k$ is satisfied for any k. Therefore, $\forall t \in T : t \in C_k$ in line 7.

In line 9, C_k is updated as $C_{k+1} \leftarrow C_k \cup T$, hence $\forall c \in C_{k+1} : c \in C(R, minlen)$.

Since both the Basis and Induction step have been proved, $\forall c \in C_k : c \in C(R, minlen)$ for any k. Therefore, the given lemma holds. □

Lemma A.1.2 *Let C be the output of algorithm 1, then*

$$\forall c \in C(R, minlen) : c \in C$$

Proof.

In algorithm 1, $\forall r \in R' : r \in C_k$ for any step since $C_0 = R'$. Hence, in line 7,

$$\forall a \in A_k, \forall r \in R' : (|a \cap r| \geq minlen \rightarrow a \cap r \in T)$$

for step k. In addition,

$$\forall r \in R', \forall r' \in R' : \left(\left|r \cap r'\right| \geq minlen \rightarrow r \cap r' \in C_k\right)$$

holds since $A_0 = R'$.

Likewise, for any $c \in C_k$ and $r \in R'$, $c \cap r$ is calculated in line 7 and added to A_{k+1} and C_{k+1} in line 8 and 9. Therefore,

$$\forall I \subseteq \{1, 2, \ldots, n\} : \left(\left|\bigcap_{i \in I} r_i\right| \geq minlen \rightarrow \bigcap_{i \in I} r_i \in C_k\right)$$

Therefore, the given lemma holds. □

According to lemma A.1.1 and lemma A.1.2, $C = C(R, minlen)$. Therefore, the output of algorithm 1 is equal to $C(R, minlen)$ in Eq. (1).

Hiroyuki Shindo received his B.E. and M.E. from Waseda University, Japan, in 2007, and 2009, respectively. In 2009, he joined NTT Communication Science Laboratories. His research interests include machine learning and natural language processing. He received the best paper award from ACL in 2012.

Tsutomu Hirao received his B.E. from Kansai University in 1995, M.E. and Ph.D. in Engineering from Nara Institute of Science and Technology in 1997 and 2002, respectively. He is currently with NTT Communication Science Laboratories. His current research interests include Natural Language Processing and Machine Learning.

Jun Suzuki received his B.E. and M.Eng. from Keio University, Japan, in 1999, and 2001, respectively and Ph.D. in engineering from Graduate School of Information Science, Nara Institute of Science and Technology, Japan, in 2005. In 2001, he joined NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corp. (NTT). His research interests include machine learning and natural language processing.

Akinori Fujino received his B.E. and M.E. degrees in precision engineering from Kyoto University, Kyoto, Japan, in 1995 and 1997, respectively. He received Ph.D. degree in informatics from Kyoto University in 2009. In 1997, he joined NTT Basic Research Laboratories, Kanagawa, Japan. In 2003, he joined NTT Communication Science Laboratories, Kyoto, Japan. His current research interests are machine learning and information extraction from complex data.

Masaaki Nagata received his B.E., M.E., and Ph.D. degrees in information science from Kyoto University, Kyoto, in 1985, 1987, and 1999, respectively. He joined NTT in 1987. He was with ATR Interpreting Telephony Research Laboratories from 1989 to 1993. His research interests include natural language processing, especially morphological analysis, named entity recognition, parsing, and machine translation.

(Editor in Charge: *Yoshihisa Udagawa*)