Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

IIS-Mine: A new efficient method for mining frequent itemsets

Supatra Sahaphong* **and Veera Boonjing**

Department of Mathematics and Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

* Corresponding author, e-mail: supatra@ru.ac.th

Received: 11 September 2011 / Accepted: 4 March 2012 / Published: 23 April 2012

Abstract: A new approach to mine all frequent itemsets from a transaction database is proposed. The main features of this paper are as follows: (1) the proposed algorithm performs database scanning only once to construct a data structure called an inverted index structure (IIS); (2) the change in the minimum support threshold is not affected by this structure, and as a result, a rescan of the database is not required; and (3) the proposed mining algorithm, IIS-Mine, uses an efficient property of an extendable itemset, which reduces the recursiveness of mining steps without generating candidate itemsets, allowing frequent itemsets to be found quickly. We have provided definitions, examples, and a theorem, the completeness and correctness of which is shown by mathematical proof. We present experiments in which the run time, memory consumption and scalability are tested in comparison with a frequent-pattern (FP) growth algorithm when the minimum support threshold is varied. Both algorithms are evaluated by applying them to synthetics and real-world datasets. The experimental results demonstrate that IIS-Mine provides better performance than FP-growth in terms of run time and space consumption and is effective when used on dense datasets.

Keywords: association rule mining, data mining, frequent itemsets mining, frequent patterns mining, knowledge discovering

INTRODUCTION

The objective of frequent itemset mining is to identify all frequently occurring itemsets using a support threshold. Decision-makers are interested in all itemsets associated with high frequencies. Association rule mining algorithms can be broken down into two major phases. The first phase finds all of the itemsets that satisfy the minimum support threshold, which are the frequent itemsets. The

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

second phase is rule generation, in which all the high confidence rules from the frequent itemsets found in the previous phase are extracted [1]. Many previous investigations focused on the first phase. Early algorithms based on generated and tested candidate itemsets have two major defects. First, the database must be scanned multiple times to generate candidate itemsets, which increases the I/O load and is time-consuming. The search space of itemsets that must be explored grows exponentially. Second, enormous candidate itemsets are generated and calculated from their supports, which consumes a large amount of CPU time [2].

To overcome the above-mentioned problems, a next generation of algorithms using a compact tree structure was proposed, called a frequent-pattern (FP) tree [3], which finds frequent itemsets directly from the data structure. However, most of the FP-tree-based algorithms have the following weaknesses. First, the mining of frequent itemsets from the FP-tree to generate a huge conditional FP-tree requires a large amount of run time and space. The best case is when a database has the same set of transactions; an FP-tree then contains only a single branch of nodes. The worst case is when a database has a unique set of transactions [3]. Second, when the users change to a new minimum support threshold for their new decision, the algorithm restarts the whole operation and scans the database twice.

Many researchers have tried to solve the above problems using a vertical data layout. However, most of the algorithms have the drawback of increasing the run time and space consumption due to the following reasons. First, when the users change to a new minimum support threshold for their new decision, the algorithm restarts the whole operation more than one time to scan a database and construct their data structure. Rescanning the database for a new minimum support threshold wastes both run time and space. Second, all of the FP-tree-based algorithms generate a huge conditional FP-tree, which has a large number of recursive processing steps and requires a large amount of run time and space consumption.

In this paper we present a new, efficient method to solve the above-mentioned problems by proposing both a data structure and a mining algorithm for decreasing the consumption of run time and space. First, the proposed method performs database scanning to construct a data structure called an inverted index structure (IIS) only once. In addition, changing the minimum support threshold does not affect the IIS; therefore, database rescanning is not required. Second, IIS-Mine is a new algorithm that mines all of the frequent itemsets without generating candidate itemsets and uses a new tree structure called the IIS_{item}Tree. IIS-Mine employs an efficient property of the extendable itemset, which decreases the number of recursive processing steps when mining frequent itemsets. The completeness and correctness of the algorithm is proved using a mathematical proof. Last, the efficiency of IIS-Mine is compared with that of FP-growth in terms of run time and space consumption through simulation experiments. Our experiments show that IIS-Mine is more efficient than FP-growth in run time and space consumption for dense datasets.

RELATED WORK

The first algorithm to generate all frequent itemsets is the AIS algorithm, which was first introduced by Agrawal et al [4]. However, this algorithm constructs a list of all of the possible

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

itemsets at each level of traversal, so infrequent itemsets that are not needed are also generated. Later, the algorithm was improved upon and renamed the Apriori algorithm by Agrawal et al [5]. The Apriori algorithm uses a level-wise and breadth-first search approach for generating association rules. Many efficient association mining techniques have been developed based on the Apriori algorithm. Vu et al. [6] proposed a rule-based location prediction technique to predict the user's featured location, but this proposal generates more candidate itemsets than are required. These algorithms are also expensive in terms of I/O load and run time when the database must be scanned multiple times to generate candidate itemsets.

The above-mentioned problems can be improved upon by using a compact tree structure and finding frequent itemsets directly from the data structure. The algorithms scan a database twice. The first scan of the database is to discard infrequent itemsets; the second is to construct a tree. The FP-growth algorithm, developed by Han et al. [7], is the most popular method. It performs a depth-first search approach in a search space. It encodes a dataset using a compact data structure called an FP-tree or prefix tree and extracts frequent patterns directly from the FP-tree. Many approaches have been proposed to extend and improve upon this algorithm. Pei et al. [8] developed the H-mine algorithm using array- and tree-based data structures to improve the main memory cost. The PatriciaMine algorithm [9] compressed Patricia tries to store datasets, which is space efficient for both dense and sparse datasets. The FP-growth algorithm [10] reduces the FP-tree traversal time using an array technique. Zhu [11] proposed a new method to compress a large database into an FP-tree with a children table but not a header table, and applied a depth-first search with this tree for the mining step, which reduces both the run time and the space consumption. Sahaphong and Boonjing [12] proposed a new algorithm which constructs a pattern base using a new method that is different from the pattern base in the FP-growth and mined frequent itemsets using a new combination method without the recursive construction of a conditional FP-tree. An approach based on the FP-tree and co-occurrence frequent items (COFI) was proposed to find frequent items in multilevel concept hierarchy by using a non-recursive mining process [13]. A new data structure called improved FP tree was proposed, which can reduce space consumption and enhance the efficiency of an attribute reduction algorithm [14]. To maintain the anti-monotone property of approximate weighted frequent patterns, a robust concept was proposed to relax the requirement for exact equality between the weighted supports of patterns and a minimum threshold [15]. However, most of the FP-tree-based algorithms require a large amount of run time and space to generate the huge conditional FP-trees. Moreover, the algorithm restarts the whole operation and requires that a database be scanned twice when the minimum support threshold is changed.

As mentioned above, most of the algorithms that mine frequent itemsets use a horizontal data layout. However, many researchers use a vertical data layout. The Eclat algorithm was proposed [16] to generate all frequent itemsets in a breadth-first search using the joining step from the Apriori property when no candidate items can be found. The Eclat algorithm is very efficient for large itemsets but is less efficient for small ones. The diffset technique [17] was introduced to improve the memory requirement. Chai et al. [18] detailed a data structure called large-item bipartite graph to accommodate the data when a database is scanned. Similar to the FP-growth algorithm, this method

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

mines frequent patterns using the recursive conditional FP-tree. The BitTableFI algorithm [19] uses horizontal and vertical data layouts to compress a database. Yen [20] presented an algorithm based on an undirected itemset graph that finds frequent itemsets by searching undirected graphs. When the database and minimum support change, this algorithm requires that the graph structure be re-searched to generate new frequent itemsets. The Index-BitTableFI [21] was developed to reduce the cost of candidate generation and to support counting. Sahaphong and Boonjing [22] proposed a new algorithm that reduces the run time. The drawback of this algorithm is its large memory consumption from generation of many repeated nodes. The JoinFI-Mine algorithm [23] uses a sorted-list structure constructed from the vertical data layout and finds all frequent itemsets using a depth-first search for joining frequent itemsets. Therefore, this algorithm consumes time and space in its joining step.

METHODS

Frequent-Itemsets Mining Problem

We introduce the basic concepts of mining frequent itemsets. All terminologies in this section are proposed by Han et al [2].

Let $I = \{x_1, x_2, ..., x_m\}$ be a set of items and $DB = \{T_1, T_2, ..., T_n\}$ be a transaction database, where $T_1, T_2, ..., T_n$ are transactions that contain items in *I*. The support, or *supp* (occurrence frequency), of a pattern *A*, where *A* is a set of items, is the number of transactions containing *A* in DB. A pattern *A* is frequent if *A*'s support is no less than a predefined minimum support threshold, *minsup*.

Given a DB and a minimum support threshold *minsup*, the problem of finding a complete set of frequent itemsets is called the frequent-itemsets mining problem.

For a greater understanding, we provide an example to illustrate the above definitions.

Example 1. An example of the database by Han et al. [2] is used here. Table 1 is a DB. It consists of 5 transactions (T_1, T_2, T_3, T_4, and T_5) and 17 items (*a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p*, and *s*). For example, the first transaction is T_1, which contains *f, a, c, d, g, i, m*, and *p*.

Table 1. A transaction database

Transaction	Item
T_1	*f, a, c, d, g, i, m, p*
T_2	*a, b, c, f, l, m, o*
T_3	*b, f, h, j, o*
T_4	*b, c, k, s, p*
T_5	*a, f, c,* e, *l, p, m, n*

IIS: Design and Construction

We present a data structure that contains transaction data called an inverted index structure (IIS). The IIS is a structure that holds a relationship between items and the transactions included within. The IIS is constructed from one scan of the DB. This original IIS can support every *minsup*;

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

therefore, it does not need to rescan the DB when the *minsup* is changed. According to the definitions in the previous section, we present a new definition with an example and an algorithm to demonstrate how to construct this IIS.

Definition 1 (IIS). Let DB be a transaction database and I be a non-empty finite set of all items in each transaction in the DB, where each transaction is a set of items in I associated with an identifier, and let S be a set of all non-empty subsets of DB. An IIS is the function $f: I \rightarrow S$ defined by $f(a) = S_1$ if T contains a for each $T \in S_1$. This function can be identified as a table consisting of the attributes of the items in I and the corresponding transactions in the DB. That is, each row in the IIS contains an item in I as well as the transactions in the DB that contain that item. The set of transactions are written in the order of their ascending identification numbers.

With the above definition, the IIS represents the relationship between each item in I and its corresponding transactions; therefore, the IIS can apply to all minimum support thresholds, and a rescan of the database is not required. We demonstrate the steps to construct the IIS through the following example.

Example 2. We use the example of a DB in Table 1. The DB is scanned once to create the IIS. The scan of the first transaction is T_1, which consists of items *f, a, c, d, g, i, m* and *p*. The transaction T_1 will be inserted for each corresponding item sorted in ascending order (*a, c, d, f, g, i, m, p*). T_1 will be the first transaction inserted in the transactions of item *a*. The second examined item is *c*, so we insert T_1 in item *c*. Next, we examine item *d*; we then subsequently insert T_1 in item *d*. The remaining items (*f*, *g*, *i*, *m* and *p*) in T_1 can be similarly inserted. The remaining transactions (T_2, T_3, T_4 and T_5) in the DB are performed in a similar manner.

Algorithm 1 shows how to construct the IIS. Figure 1 shows all of the items of the IIS after scanning the DB once, and the bold items are all frequent items that have a support greater than or equal to the *minsup*, which is assumed to be 3.

```
Algorithm 1 (IIS construction)
Input: DB.
Output: IIS.
Method: The IIS is constructed as follows.
1 Begin
2   Create header that contains all items.
3   For each transaction T in DB do  // scanning DB once
4       Sort items in T                    // ascending order
5       Create transaction to each corresponding item
6   End //For
7 End //Begin
```

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Item	Transaction			
a	T_1	T_2	T_5	
b	T_2	T_3	T_4	
c	T_1	T_2	T_4	T_5
d	T_1			
e	T_5			
f	T_1	T_2	T_3	T_5
g	T_1			
h	T_3			
i	T_1			
j	T_3			
k	T_4			
l	T_2	T_5		
m	T_1	T_2	T_5	
n	T_5			
o	T_2	T_3		
p	T_1	T_4	T_5	
s	T_4			

Figure 1. An example of the IIS

IIS-Mine Algorithm

We present a new algorithm called IIS-Mine. This algorithm uses a new tree structure, called the IIS_{item}Tree, to mine frequent itemsets. The main features of this algorithm are as follows: (1) every frequent itemset is found without generating candidate itemsets; (2) the algorithm reduces the recursion of mining steps using the property of extendable itemset; and (3) the algorithm supports the mining of frequent itemsets with any value of the *minsup* without needing to rescan the database. From the above features, we can quickly find the frequent itemsets and completely and correctly obtain them. We now introduce the terminologies of the IIS_{item}Tree, its construct, the theorem, the examples and the algorithms to describe how to mine frequent itemsets.

Definition 2 (Itemset-tree structure). An itemset-tree structure is a tree structure constructed from the IIS. It is a finite set of one or more nodes with the following structure:

(i) It consists of the root which contains an item, a set of item subtrees as the children of the root, and a set of header tables.

(ii) Each node in this tree comprises five fields: *item-name*, which registers which item this node represents; *support,* which registers the number of transactions represented by the portion of the path reaching this node; *same-item,* which represents a pointer that points to the node in the itemset-tree structure that carries the same item-name; *parent*, which represents a pointer that points to the previous node in the same path; and *child*, which represents a pointer that points to the child node.

(iii) Each member of the *header table* consists of two fields, *item-name* and *head of node link*, where *head of node link* represents a pointer that points to the first node in the itemset-tree structure carrying the *item-name*.

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Definition 3 (IIS_{item}tree). Let x_0 be an arbitrary frequent item in a given transaction database and IIS be the inverted index structure of the transaction database. A tree T constructed from the IIS is called an inverted index structure- x_0 tree, denoted by $IIS_{x_0}tree$, if it satisfies the following:

(i) Each node of T is of the form $(A:s)$, where A is a frequent itemset and s is its support. If $(A:s)$ is a node of T and $A=\{a\}$, where a is a frequent item, then $(A:s)$ is simply written by $(a:s)$.

(ii) Let $(x_0:s_0)$ be its root, where s_0 is the support of x_0.

(iii) Let $x_0,x_1,...,x_k$ be frequent items in the IIS, and $s_i = supp\{x_0,x_1,...,x_i\}$ for all $i=0,1,...,k$. In this case, $P=((x_0:s_0),(x_1:s_1),...,(x_k:s_k))$ is a path from the root $(x_0:s_0)$ to a leaf $(x_k:s_k)$ of the tree T if and only if $s_0 \geq s_1 ... \geq s_k > 0$, $x_0 <_l x_1 <_l ... <_l x_k$, where $<_l$ is the lexicographic order. If a is a frequent item in the IIS and if $(a:s)$ is a node of T such that $x_k < a$ or $x_i <_l a <_l x_{i+1}$ for all $i=0,1,...,k$, then $supp\{x_0,x_1,...,x_i,a\}=0$ and $(a:s)$ is not a node of P.

The header table of $IIS_{x_0}tree$ is a set of all frequent items a of a node $(a:s)$ of this tree.

Based on the above definition, we have the IIS_{item}Tree construction algorithm, as shown in Algorithm 2. It is evident that if *minsup*>0 is a minimum support threshold, then every frequent itemset can be derived from an IIS_{item}Tree.

```
Algorithm 2 (IIS_item Tree construction)
Input: IIS.
Output: IIS_item Tree.
Method: An IIS_item Tree is constructed as follows.
1Begin
2  Create header table
3  Read frequent item x in IIS
4  Create root R and initial supp(R) to 1
5  Link R to header table
6  For each transaction T of root R where T = T_1 to T_n do
7     While next frequent item≠(last frequent item)+1 do
8        Read next frequent item (N) that has same T with R
9        Call InsertTree (N,R)
10    End//While
11 End//For
12End //Begin
Procedure InsertTree (N,R)
1Begin
2  If IIS_R Tree has a node C such that C.item-name = N.item- name then
3     Increment supp(N) by 1
4  Else
5     Create new node N and initial supp(N) to 1
6     Link N to N's parent
7     Link N to N's header table
8     Link N to same-item
9  End //If
10End //Begin
```

If no confusion arises, then $\{x_1,x_2,...,x_n\}$ and $(\{x_1,x_2,...,x_n\}:s)$ are replaced by $x_1x_2...x_n$, and $<x_1x_2...x_n:s>$ respectively, where $x_1,x_2,...,x_n$ are items.

Example 3. In this example, we describe the steps to construct all of the IIS_{item}Trees except the last frequent item p using the IIS in Figure 1 and *minsup* = 3. The first frequent item in the IIS is

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

a; therefore, we first construct the *IIS$_a$Tree*, and item *a* is a root. We obtain two paths: (<*a*:2>, <*c*:2>, <*f*:2>, <*m*:2>, <*p*:2>) and (<*a*:1>, <*b*:1>, <*c*:1>, <*f*:1>, <*m*:1>). The first path consists of frequent items (*a*,*c*,*f*,*m*,*p*) that appear twice in the DB. Similarly, the second path indicates frequent items (*a*,*b*,*c*,*f*,*m*) that are contained in only one transaction in the DB. These two paths share the frequent item *a*; thus, *a* appears three times in the DB. Other IIS$_{item}$Trees, except the *IIS$_p$Tree*, can be similarly constructed. In Figure 2, all of the IIS$_{item}$Trees, except the last *IIS$_p$Tree*, are illustrated.

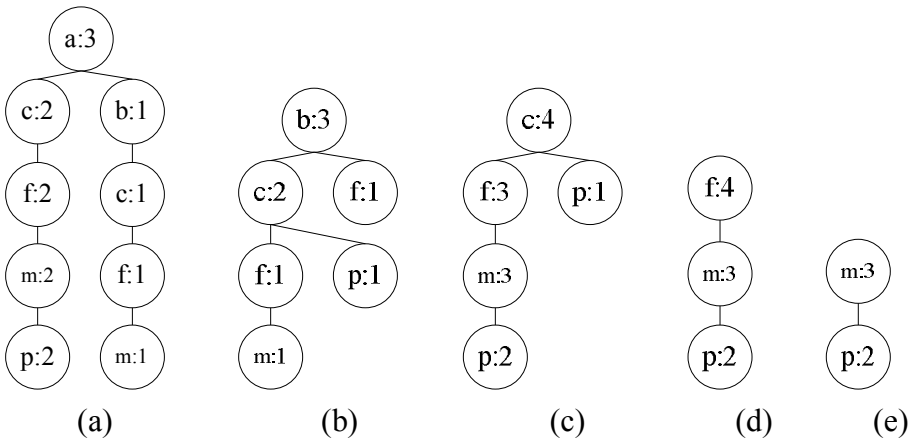


Figure 2. IIS$_{item}$Trees

Definition 4 (A$_1$-tree). Let DB be a transaction database; let *T* be a tree such that each node of the tree is of the form $(A:s)$, where *A* is a frequent itemset in DB and *s* is the support of *A*; and let A_1 be a frequent itemset in the DB. *T* is called an *A*$_1$*-tree* if, for any path *P* of *T*, *P* is of the form $P=((A_1:s_1),(A_2:s_2),...,(A_k:s_k))$, where *k* is a positive integer; A_i and A_j are pairwise disjoint frequent itemsets for all $i, j = 1,2,\ldots,k$ with $i \neq j$; s_i is the support of $\bigcup_{m=1}^{i} A_m$ for $i = 1,2,\ldots,k$; (A_1:s_1) is the root of *T*; and $(A_k:s_k)$ is a leaf of *T*.

Example 4. According to Figure 2 (a), the *ac*-tree is shown in Figure 3.

Definition 5 (Prefix subpath). Let *T* be a tree, a_1 be the root of *T*, and $P=(a_1,...,a_m)$ be a path of *T*, where *m* is a positive integer. Every path $Q=(a_1,...,a_k)$ of *T* is then called a prefix subpath of *P*, where $1 \leq k \leq m$.

Example 5. According to Figure 2(a), the paths ((*a*:3),(*c*:2)) and ((*a*:3),(*c*:2),(*f*:2)) are prefix subpaths of ((*a*:3),(*c*:2),(*f*:2),(*m*:2)).

Definition 6 (Subheader). Let *T* be an *A-tree* and (*x*:*s*(*x*)) be a node of *T*, where *x* is a frequent item in a given transaction database and *s*(*x*) is the support of *x*. Suppose that all of the nodes (of *T*) containing *x* are only in the paths $P_1,\ldots,P_k$ of *T* from the root (*A*:*s*) to some leafs of *T*, and suppose that Q_i is a prefix subpath (of P_i) from the root $(A:s)$ to $(x:s(Q_i))$ for all $i=1,...,k$. The subheader of *T*, denoted by *SH*(*A*), is defined as the order set *SH(A)* = {*x*|*x* is a frequent item not contained in *A*, $(x:s(Q_i))$ is a node in Q_i for $i = 1,\ldots,k$ and $\sum_{i=1}^{k} s(Q_i) \geq minsup\}$.

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Example 6. According to Figure 2(a), SH(*a*) = {*c*,*f*,*m*}, where SH(*a*) is the subheader of the tree in Figure 2 (a), *supp*(*c*) = 3, *supp*(*f*) = 3, and *supp*(*m*) = 3.

Definition 7 (Conditional itemset-tree). Let A_1 be a frequent itemset, T be an A_1-*tree*, x_2 be a frequent item with $x_2 \in SH(A_1) - A_1$, and $A_1x_2 := A_1 \cup \{x_2\}$ be a frequent itemset. A *conditional* A_1x_2-*tree*, denoted by $T(A_1x_2)$, is a tree that satisfies the following:

(i) $(A_1x_2 : s_2)$ is the root of $T(A_1x_2)$, where s_2 is the support of A_1x_2 and $s_2 \geq minsup$.

(ii) The number of items in $SH(A_1) > 1$.

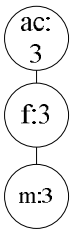
(iii) All of the paths are derived from T in the following way: $Q = ((A_1x_2 : s_2),(x_3 : s_3),...,(x_k : s_k))$ is a path of $T(A_1x_2)$ if and only if a path $P = ((A_1 : s_1),(x_2 : s_2),...,(x_l : s_l))$ exists from the root $(A_1 : s_1)$ to the leaf $(x_l : s_l)$ of T, where $l \geq k$ and x_k is in $SH(A_1)$; and if there is a node $(x_r : s_r)$ of P such that $r > k$ and $((A_1 : s_1),(x_2 : s_2),...,(x_r : s_r))$ is a prefix subpath of P, then x_r must not be in $SH(A_1)$.

Example 7. According to Figure 2 (a), the conditional itemset-tree, or conditional *ac*-tree, is shown in Figure 4.

Notably, every non-empty subset of a frequent itemset is also frequent. This fact leads to the following definition.

Definition 8 (Frequent itemset* of length m derived from tree). Let A be a frequent itemset, x be a frequent item, and $x \notin A$. Suppose that T is a conditional Ax-tree, m is a positive integer greater than 1, and $Ax = A \cup \{x\}$ has m elements. Ax is called a frequent itemset* of length m derived from T, denoted by $FS^*_m(Ax)$, if T contains precisely two nodes and Ax is a frequent itemset, or if $SH(A) = \{x\}$.

Example 8. According to Figure 5, the frequent itemset* of length 4 derived from the conditional *acf*-tree is $FS^*_4(acfm) = acfm.$.



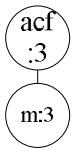


Figure 3. *ac*-tree **Figure 4.** Conditional *ac*-tree **Figure 5.** Conditional *acf*-tree

Definition 9 (Extendable frequent itemset*). Let m be a positive integer greater than 1, T be a $T(Ax)$ defined as in definition 7 with $A_1 = A$ and $x_1 = x$, and Ax be a frequent itemset* of length m derived from T. Each itemset in $FS^*_m(Ax)$ is said to be extendable if $m \geq 3$. For every k = 2, 3,…, m, we let $Ext^*_k(Ax)$ denote the set of all itemsets containing exactly k items of $FS^*_m(Ax)$, where $m \geq 3$. Each element of $Ext^*_k(Ax)$ is called an extendable frequent itemset* (derived from T) of length k for all $k \geq 2$. The set of all extendable frequent itemsets* of length k that is denoted by $Ext^*_k = \cup\{Ext^*_k(Ay) \mid Ay$ is a frequent itemset* of length at least k derived from a conditional Ay-*tree*$\}$ for all $k \geq 2$.

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Example 9. According to the previous example, the length of $FS_4^*(acfm)$ is 4; we then find that $Ext_2^*(acfm)$ = {*ac*, *af*, *am*, *cf*, *cm*, *fm*} and $Ext_3^*(acfm)$ = {*acf*, *acm*, *afm*, *cfm*}. Therefore, Ext_2^* = {*ac*, *af*, *am*, *cf*, *cm*, *fm*, and all members of other $Ext_2^*(Ay)$ } and Ext_3^* = {*acf*, *acm*, *afm*, *cfm*, and all members of other $Ext_3^*(Ay)$ }.

Definition 10 (Frequent itemset* of length *m*). Let *k* be the maximum length of all frequent itemsets in a transaction database, $FS_m^*(Ax)$ and Ext_k^* be given as in definitions 8 and 9 respectively; let $A_m = \{FS_m^*(Ax) \mid Ax$ being a frequent itemset* of length *m* derived from *T*} for *m* = 2, 3,…, *k*; define $FS_2^* = A_2 \cup \{Ax \mid Ax$ being a frequent itemset of length 2}; and define $FS_m^* = A_m$ for *m* = 2, 3,…, *k*}. Let FI_m^* be defined by $FI_1^* = \{\{x\} \mid x$ being a frequent item} and $FI_m^* = FS_m^* \cup Ext_m^*$ for *m* = 2, 3,…, *k*}. Any element of FI_m^* is called the frequent itemset* of length *m*.

Example 10. According to the previous example, we find that FI_2^*= {*ac*, *af*, *am*, *cf*, *cm*, *fm*}, FI_3^*= {*acf*, *acm*, *afm*, *cfm*}, and FI_4^*= {*acfm*}.

Definition 11 (Frequent itemset*). Let FI_m^* be given as in definition 10, and let FI^* denote $\cup_{m=1}^{k} FI_m^*$, where *k* is the maximum length of all frequent itemsets in a transaction database. Any element of FI^* is called a frequent itemset*.

Example 11. According to the previous example, FI^* is {*ac*, *af*, *am*, *cf*, *cm*, *fm*, *acf*, *acm*, *afm*, *cfm*, *acfm*}.

On the basis of the above definitions and examples, Algorithm 3 presents the IIS-Mine algorithm to show how it can be used to mine all frequent itemsets.

```
Algorithm 3: (IIS-Mine: Mining frequent itemsets using IIS_itemTree)
Input: IIS, IIS_item Trees constructed according to Algorithm 2, and minsup
Output: FI*
Procedure AllFreqItemset (IIS, FI*)
1 Begin
2   For each frequent item x in the IIS do
3     FI_1^* = {{x | x ∈ I, supp(x) ≥ minsup}
4     Call IIS_xTree , which is constructed from Algorithm 2
5     If Tree ≠{}
6       Call IIS-Mine (Tree, x)
7     End //If
8   End //For
9   Find FI^* = ∪_{m=1}^{k} FI_m^*                    // FI* is given in definition 11
10 End //Begin
Procedure IIS-Mine(Tree, x)
1 Begin
2   Call SubHeader (A-tree, subheaderA)
3   Generate all Ax with its support
     //All Ax are the frequent itemsets where A is the root of A-tree and x ∈subheaderA
4   For each |λ|>1 do  // λ is Ax
5    Flag=1
6    While Flag = 1 do
7      Call SkipFreqItemset (subheaderA, λ , δ ,FlagRepeat, FlagTree)
8      If FlagRepeat=0 and FlagTree=0 then  // (δ ∉ FI_m^* )
9      Call CondItemsetTree (A-tree, δ , conditional δ -tree)
```

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

```
10    Else
11      Flag=0
12    End // If
13    If conditional Ax-tree contains greater than two nodes
14       Call IIS-Mine(conditional δ -tree, x)
15    Else Flag=0
16    End //If
17  End //While
18  If | FS*m(δ) |≥3 and FS*m(δ) ∉ FI*m then
19    Call ExtFreqItemset( FS*m(δ) , Ext*k )                    // FS*m is given in definition 8
20    Save FS*m(δ) and all Ext*k to FI*m                        // FI*m is given in definition 10
21  Else
22     If FS*m(δ) ∉ FI*m then
23       Save FS*m(δ) to FI*m
24     End //If
25  End //If
26  End //For
27 End //Begin
Procedure SubHeader (A-tree, subheaderA)
1 Begin
2  Each frequent item x of A-tree                              // SH(A) is given in definition 6
3  Find {(x:s(x))|x ∈SH(A), s(x) is the support of x in A-tree}
4 End // Begin
Procedure CondItemsetTree(Tree, δ , conditional δ -tree)
1 Begin
2  Scan tree once to collect the paths that have an association with root δ
3  For all paths are derived from tree do
4    Connect all paths to δ
5  End //For
6 End //Begin
Procedure SkipFreqItemset (SubheaderA, λ , δ , FlagRepeat, FlagTree)
1 Begin // To skip the construction of conditional item tree
2 // x is the frequent item in subheaderA  // λ is the root of tree; or a frequent itemset
3 // FlagRepeat=0 means δ ∉ FI*m          // FlagTree=0 means the conditional item-tree is constructed
4 // n is the maximum number of elements of itemsets in subheaderA
5  FlagRepeat =1, FlagTree=1
6  β = λ, α = β
7  While( FlagRepeat=1 and (order of x ≤n )) do
8    If β ∉ FI*m then
9       FlagRepeat=0, FlagTree=0
10       If x is the last item
11         If | δ |= 2 then FlagTree=1
12         End //If
13         δ = α
14       End //If
15      Else
16       If x is the last item then
17         Increment order of item x
18       Else
19         Increment order of item x
20         β = δ ∪x
21         α = δ
22       End// If
```

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

23 δ = β
24 End //If
25 End //while
26 End //Begin
Procedure ExtFreqItemset($FS_m^*(\delta)$, Ext_k^*)
1 Begin // Ext_k^* is given in definition 9
2 Generate all subsets of $FS_m^*(\delta)$ and save to Ext_k^*
3 End // Begin

Example 12. This example is given to demonstrate how the proposed IIS-Mine algorithm can be used to mine all frequent itemsets. Assume *minsup* = 3 for all of the definitions above, and for Examples 1-3, Figure 2 and Algorithm 3. For simplicity, this example is divided into five main steps ordered by five frequent items in the IIS. The proposed mining algorithm proceeds as follows.

Let step 1 be the first main step. According to Procedure AllFreqItemset, the frequent item *a* is the first frequent item in the IIS that is mined. The *IIS$_a$Tree* is constructed, as illustrated in Figure 2(a). The algorithm calls Procedure IIS-Mine, so Procedure Subheader is called in order. The SH(*a*) is (*c*,*f*,*m*), where *supp*(*c*) = 3, *supp*(*f*) = 3 and *supp*(*m*) = 3. After line 3 in the procedure, IIS-Mine generates all of the frequent itemsets with its support, i.e. *ac*, *af* and *am*; all of these frequent itemsets have support equal to three. Let steps 1.1, 1.2 and 1.3 represent each of the frequent itemsets.

The step 1.1, the first frequent itemset is *ac*, which has support equal to three. The algorithm checks |*ac*|>1 and then calls Procedure SkipFreqItem. At this procedure, *ac* is not in FI_2^*, so the algorithm rolls back to line 9 of Procedure IIS-Mine. The frequent itemset *ac* with its support is defined to be the root; then, the conditional *ac*-tree, which has root "<*ac*:3>", is constructed using the input *IIS$_a$Tree*. The conditional *ac*-tree is illustrated in Figure 4. The algorithm is iterated by calling Procedure IIS-Mine again because the conditional *ac*-tree contains more than two nodes; let this call be step 1.1.1.

In step 1.1.1, the Procedure IIS-Mine is called; SH(*ac*) = (*f*,*m*), where *supp*(*f*) and *supp*(*m*) are then equal to 3. At line 3, the algorithm generates frequent itemsets, which are *acf* and *acm*, where *supp*(*acf*) and *supp*(*acm*) are then equal to 3. The first frequent itemset in this step is *acf* and $|acf|>1$, so the procedure SkipFreqItem in line 7 is processed, and it finds that *acf* is not in FI_3^*. Next, the algorithm rolls back to line 9 to construct a conditional *acf*-tree that has a conditional *ac*-tree as an input tree, which is illustrated in Figure 5. The condition of line 13 is that the conditional *acf*-tree contains only two nodes, so the algorithm obtains $FS_4^*(acfm) = acfm$. At line 18, the size of $FS_4^*(acfm)$ is greater than three and $FS_4^*(acfm) \notin FI_4^*$. Thus, at line 19, Procedure ExtFreqItemset is called to find all of the subsets of $FS_4^*(acfm)$, which are $Ext_2^*(acfm)$ and $Ext_3^*(acfm)$: $Ext_2^*(acfm)$ = {*ac*, *af*, *am*, *cf*, *cm*, *fm*} and $Ext_3^*(acfm)$ = {*acf*, *acm*, *afm*, *cfm*}; hence, FI_2^* = {*ac*, *af*, *am*, *cf*, *cm*, *fm*}, FI_3^* = {*acf*, *acm*, *afm*, *cfm*}, and FI_4^* = {*acfm*}.

In step 1.1.2, the next frequent itemset generated together with step 1.1.1 is *acm*. At line 7 of Procedure IIS-Mine, the algorithm calls Procedure SkipFreqItem and obtains *acm*, which is already a

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

member of FI_3^*; *m* is the last frequent item in SH(*ac*), so we exit from this step without the construction of a conditional *acm*-tree.

In step 1.2, at line 4 of Procedure IIS-Mine, the next frequent itemset is *af*, and at line 7, Procedure SkipFreqItem is called and obtains *af*, which is contained in FI_2^*. In the loop in line 20 of Procedure SkipFreqItem, after *af* is combined with the next frequent item in SH(*a*), which is *m*, *afm* is obtained, which is contained in FI_3^*, and item *m* is the last item in SH(*a*). The algorithm rolls back to line 8 of Procedure IIS-Mine, so the conditional *af*-tree is not constructed.

In step 1.3, at line 4 of Procedure IIS-Mine, the next frequent itemset is *am*. At line 7, Procedure SkipFreqItem is called and obtains *am*, which is contained in FI_2^*, and there is no frequent item in SH(*a*). Therefore, the conditional *am*-tree is not constructed.

Let step 2 be the second main step. According to line 2 of Procedure AllFreqItemset, *b* is the second frequent item in the IIS that is mined. After line 4, the *IIS$_b$Tree* is constructed, as illustrated in Figure 2(b). The algorithm calls Procedure IIS-Mine at line 6. At line 2 of Procedure IIS-Mine, Procedure Subheader is called and obtains an empty SH(*b*), so the size of the frequent itemset generated with *b* is one. The processing of this step is terminated, and we return to line 2 of Procedure AllFreqItemset.

Let the third main step be step 3. According to line 2 of Procedure AllFreqItemset, *c* is the third frequent item in IIS that is mined. Line 4 is called to construct the *IIS$_c$Tree*, as illustrated in Figure 2(c). At line 6, the algorithm calls Procedure IIS-Mine to mine frequent itemsets. At line 2 of Procedure IIS-Mine, Procedure Subheader is called to obtain the SH(*c*) that is (*f*,*m*,*p*). Next, at line 3, the algorithm generates frequent itemsets, which are *cf*, *cm* and *cp*, where *supp*(*cf*), *supp*(*cm*) and *supp*(*cp*) = 3. Let steps 3.1, 3.2 and 3.3 represent each of the frequent itemsets.

In step 3.1, at line 4 of Procedure IIS-Mine, the first frequent itemset is *cf*, where $|cf| > 1$; line 7 then calls Procedure SkipFreqItemset. At Procedure SkipFreqItemset, *cf* combines with the next frequent item in SH(*c*), which is *m*, so *cfm* with a support of 3 is obtained after processing lines 19-23. Next, line 8 is checked, and as *cfm* is already obtained in FI_3^*, lines 19-23 are checked again, and *cfmp* is obtained. The frequent itemset *cfmp* is not a member in FI_4^*, and *p* is the last frequent item in SH(*c*), so *cfm* is set to be a root for a conditional *cfm*-tree. The algorithm goes back to line 8 of Procedure IIS-Mine to construct a conditional *cfm*-tree, where the *IIS$_c$Tree* is an input tree, which is illustrated in Figure 6. *Supp*(*p*) is less than *minsup*, hence $FS_3^*(cfm) = cfm$. Procedure ExtFreqItemset in line 18 is not called because $FS_3^*(cfm)$ is contained in FI_3^*. In this step, the algorithm skips the construction of the conditional *cf*-tree.

In step 3.2, according to line 4 of Procedure IIS-Mine, the next frequent itemset is *cm*, and Procedure SkipFreqItemset in line 7 is called. Line 8 of Procedure SkipFreqItemset checks that *cm* is a member in FI_2^*. Next, lines 19-23 are checked, so *cm* combines with the next item in SH(*c*), which is *p*, and we obtain *cmp* with a support of 3. The frequent itemset *cmp* is not in FI_3^*, and *p* is the last frequent item in SH(*c*), so *cm* is set to be a root for a conditional *cm*-tree. The algorithm goes back to line 8 of Procedure IIS-Mine to construct a conditional *cm*-tree, where the *IIS$_c$Tree* is an input tree, which is illustrated in Figure 7. *Supp*(*p*) is less than *minsup*, hence $FS_2^*(cm) = cm$ and $FS_2^*(cm)$ are already members in FI_2^*.

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

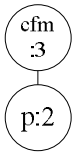


Figure 6. Conditional *cfm*-tree

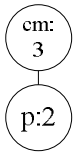


Figure 7. Conditional *cm*-tree

In step 3.3, according to line 4 of Procedure IIS-Mine, the next frequent itemset is *cp*. Next, at line 5, Procedure SkipFreqItemset is called, and *cp* is not a member in FI_2^*. There is no frequent item in SH(*c*), so this procedure is terminated, and we return to line 8 of Procedure IIS-Mine. After lines 22-23 are checked, the new answer is contained in FI_2^*, which is *cp*. This step skips the construction of a conditional *cp*-tree.

The remaining steps such as the fourth main step, which constructs the IIS_fTree and is illustrated in Figure 2(d), and the fifth main step, which constructs the IIS_mTree and is illustrated in Figure 2(e), are performed in the same way in sequence.

The complete frequent itemsets are shown by item in Table 2 and by length in Table 3.

Table 2. Complete frequent itemsets by item

Item	Frequent itemset
a	*a*, *acfm*, *ac*, *af*, *am*, *cf*, *cm*, *fm*, *acf*, *acm*, *amf*, *cfm*
b	*b*
c	*c*, *cp*
f	*f*
m	*m*
p	*p*

Table 3. Complete frequent itemsets by length

m-Length	Frequent itemset
1	*a*, *b*, *c*, *f*, *m*, *p*
2	*ac*, *af*, *am*, *cf*, *cm*, *cp*, *fm*
3	*acf*, *acm*, *afm*, *cfm*
4	*acfm*

The advantages of our algorithm are as follows. First, according to step 1.1.1 of Example 12, the frequent itemsets *acfm* are obtained, which are derived from a conditional *acf*-tree. This step shows the properties of an extendable frequent itemset*, which are given in Definition 9 and Procedure ExtFreqItemset in Algorithm 3. This step then finds all of the subsets of *acfm*, so we derive ten frequent itemsets, which are *ac*, *af*, *am*, *cf*, *cm*, *fm*, *acf*, *acm*, *amf* and *cfm*, without contributing more trees or using recursion to mine. Therefore, our algorithm can reduce many

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

subsequent steps in mining frequent itemsets. It can be noticed that if $|FS_m^*(Ax)|$ is large, then the property of an extendable frequent itemset* is frequently used. Second, the method is also good for reducing run time and space consumption, and its performance will be shown in the experimental section. Last, according to step 3.1 of Example 12, the conditional *cfm*-tree is obtained, which reduces one node in the tree because of the step that sets frequent itemsets as the root node in Procedure SkipFreqItemset in Algorithm 3. The algorithm reduces the number of nodes, levels and size of the tree, thus reducing space consumption. In general, users change many minimum support thresholds to make their decision. Our method, which uses an IIS and the restart algorithm shown in Algorithm 3, supports the ability to make these changes without rescanning the database.

Correctness

The following theorem and proof are given to demonstrate that the proposed IIS-Mine algorithm can mine frequent itemsets completely and correctly.

Theorem: The set of all frequent itemsets* derived from IIS-Mine is the complete set of all of the frequent itemsets.

Proof: Let *I* be the nonempty finite set of all items in a given transaction database, *FI* denote the set of all frequent itemsets in the transaction database, and $\alpha = minsup > 0$. It must be proved that $FI = FI^*$.

First, we prove that $FI \subseteq FI^*$. Let $F = \{a_1,...,a_k\} \in FI$ with $a_1 <_l ... <_l a_k$, and $s_i = supp\{a_1,...,a_i\}$ for all $i = 1,...,k$. Then, $s_1 \geq ... \geq s_k \geq \alpha$, and an item *b* exists such that $b \geq a_1$, and $IIS_b tree$ contains $a_1,...,a_k$. It can be assumed that b is the item in *I* having these properties because *I* is the nonempty finite set of all items. Because $s_i \geq ... \geq s_k \geq \alpha$ for all $i = 1,...,k$, there exists a path $((b_1 : s_1'),...(b_l : s_l'))$ of $IIS_b tree$ containing $(a_1 : s_1),...,(a_k : s_k)$; that is, for all $i = 1,...,k$, there exists $j = 1,...,l$ such that $(a_i : s_i) = (b_j : s_j)$. It is obvious from the IIS-Mine algorithm that $FI \subseteq FI^*$.

We also show that $FI^* \subseteq F$I. Let $F = \{a_1,...,a_k\} \in FI^*$ with $a_1 <_l ... <_l a_k$. Then, for some positive integer *m*, $F \in FI_m^*$. It is evident from definition 10 that if $m = 1$, $F \in FI_m^*$ implies $F \in FI$. Now, suppose $m \geq 2$; then, $F \in FI_m^*$ or $F \in Ext_m^*$. In the first case, $F \in FS_m^*$, and from Definition 8, $SH\{a_1,\ldots,a_{k-1}\}=(a_k)$ or the conditional $\{a_1,...,a_{k-1}\}a_k$ tree contains exactly two nodes, $(\{a_1,...,a_{k-1}\} : s_{k-1})$ and $(a_k : s_k)$, where $s_i = supp\{a_1,...,a_i\}$ for $i = k-1, k$. Then from definitions 7 and 8, we obtain $\{a_1,...,a_{k-1}\}a_k = \{a_1,...,a_k\} = F \in FI$. In the other case, suppose that $F \in Ext_m^*$ for $m \geq 2$. Then from definition 9, an *Ax* exists such that $F \in Ext_m^*(Ax)$, and *Ax* is a frequent itemset* of length at least *m* derived from the conditional *Ax*-tree. Again, from definition 9, a positive integer *k* greater than 2 exists such that *F* has itemsets containing precisely *m* items of $FS_k^*(Ax)$, and from definitions 6 and 8, $FS_k^*(Ax)$ is a frequent itemset, hence $F \in FI$. The proof is complete.

RESULTS AND DISCUSSION

We have presented the experiments in which the run time, memory consumption and scalability are tested for the IIS-Mine algorithm and FP-growth algorithm with different datasets and varying minimum support thresholds. The experiments were performed on a Microsoft Windows XP

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Professional Version 2002 Service Pack 3 operating system on a personal computer with 1 GB of main memory and Pentium (R) CPU 3.00 GHz. All algorithms were coded using C language. Two groups of benchmark datasets, i.e. two synthetic datasets and two real datasets, were used.

For the first group of datasets, we also presented the experimental results for two synthetic datasets generated by the IBM Almaden Quest research group [24-25]. The datasets serve as the FIMI repository, which is a result of the workshops on frequent itemset mining implementations [26, 27]. The two original databases of synthetic datasets are T10I4D100K and T40I10D100K, which are sparse datasets. The notation *TxIyDzK* denotes a dataset where *K* is 1,000 transactions. Table 4 lists the parameters of the synthetic datasets, which vary in the number of transactions, i.e. 20%, 40%, 60% and 80% of the original database.

Table 4. Parameters of the synthetic datasets

$\|T\|$	Average number of items per transaction
$\|I\|$	Average length of a frequent itemset
$\|D\|$	Number of transactions

For the second group of datasets, the real datasets from the UCI machine learning repository [28] were used to test the proposed method. The real datasets used in the experiment were Chess [29] and Mushroom [30], which are dense datasets with a great number of long frequent itemsets. The characteristics of the real datasets are shown in Table 5.

Table 5. Characteristics of real datasets

Real dataset	Description
Chess	Average number of items per transaction = 37, number of transactions = 3,196, and number of items = 75
Mushroom	Average number of items per transaction = 23, number of transactions = 8,124, and number of items = 119

Run Time

Figures 8(a) and 8(b) show the performance of the algorithms on two synthetic datasets, T10I4D100K and T40I10D100K respectively. In Figure 8(a), IIS-Mine performs better than FP-growth in every support threshold. The gap in the graph becomes larger as the support threshold decreases. In Figure 8(b), when the minimum support is set at 20%, 17.5%, 15% or 12.5%, the run time between the two algorithms is not very different. However, when the minimum support is set at 10%, 7.5% or 5%, the run time of FP-growth increases significantly when compared to that of IIS-Mine, which confirms that IIS-Mine performs better than FP-growth. The results shown in Figures 8(a) and 8(b) can be explained as follows. With sparse datasets, when the minimum support is high, the number of frequent itemsets is low. However, when the minimum support is low, many frequent itemsets are obtained. IIS-Mine is always faster than FP-growth method, especially when the

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

minimum support is low, because FP-growth constructs bushy and wide FP-trees when the minimum support is low. So FP-growth is computationally expensive for tree traversing the FP-trees. IIS-Mine has the step of finding the root node from previous frequent itemsets, which can reduce the number of nodes and levels of the conditional itemset-tree. Therefore, traversing in the conditional itemset-tree is on a reduced tree, which can result in a low run time consumption. However, the run time of both algorithms relies on the length of the transaction (as observed in a comparison of the graphs in Figures 8(a) and 8(b) with a minimum support of 5%); when the transaction is long, so it the run time of both algorithms.

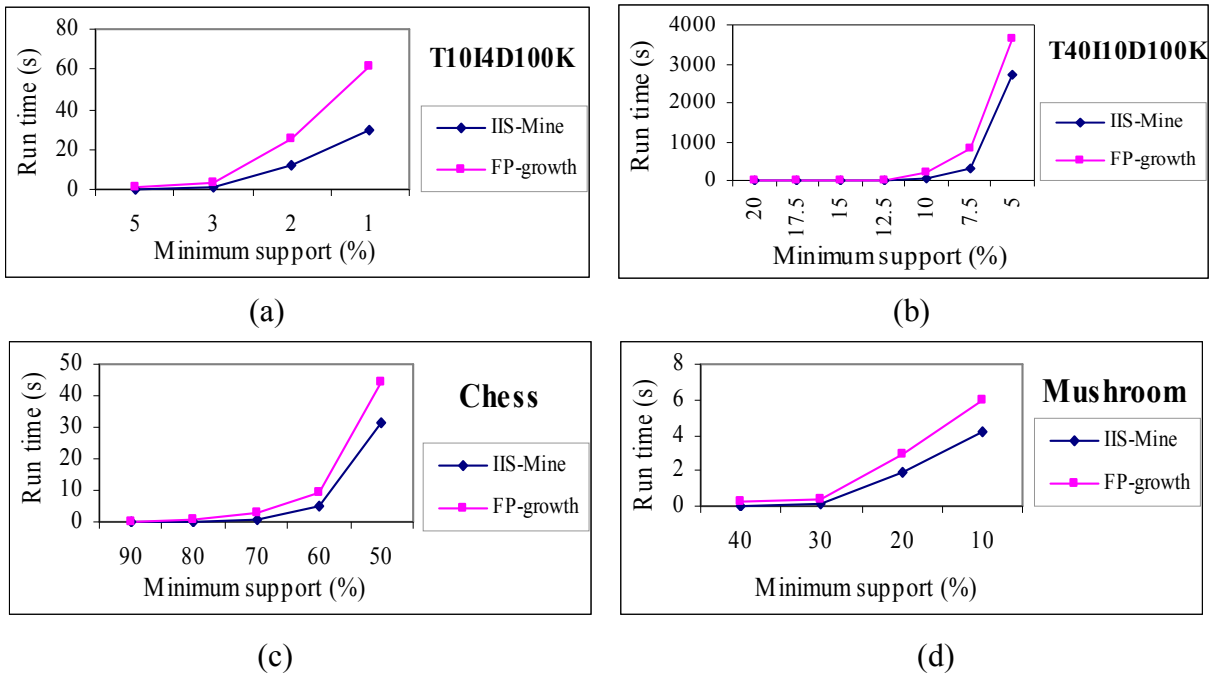


Figure 8. Run time of mining on: a) T10I4D100K; b) T10I4D100K; c) Chess; d) Mushroom

Figures 8(c) and 8(d) show the performance of algorithms on two dense datasets: chess and mushroom. Figure 8(c) shows that the run time of IIS-Mine is better than that of FP-growth in every support threshold. The run time of both algorithms increases when the minimum support threshold is reduced to 50%. In Figure 8(d), IIS-Mine again performs better than FP-growth in every minimum support. The run time of FP-growth increases significantly compared with IIS-Mine when the minimum support is less than 30%. The results shown in Figures 8(c) and 8(d) can be explained as follows. In the two Figures, IIS-Mine is faster than FP-growth for dense datasets. The main work in FP-growth is traversing FP-trees and constructing new conditional FP-trees after the first FP-tree is constructed from the original database. For dense datasets, we have found from numerous experiments that the time spent on traversing FP-trees is very long. IIS-Mine improves this problem using the property of extendable itemsets to reduce the number of recursive mining steps so that the size and number of constructing and traversing the trees are reduced. The run time of IIS-Mine is then less than that of FP-growth.

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

Memory Consumption

Figures 9(a) and 9(b) show the memory consumption of the algorithms on the synthetic datasets. In Figure 9(a), FP-growth consumes more memory than IIS-Mine. The graphs clearly separate out when the minimum support is less than 3%. In Figure 9(b), there is no difference in memory consumption until the minimum support is less than 15%. Thus, we can see that IIS-Mine consumes less memory than FP-growth on synthetic datasets. The large memory consumption of FP-growth when running on synthetic datasets can be explained by the fairly low minimum support and the presence of many single items in the datasets; therefore, FP-growth constructs wide and bushy trees to mine all frequent itemsets. However, IIS-Mine uses the property of extendable itemsets, which reduces the construction of conditional itemset-trees, and uses the step of finding the root node from previous frequent itemsets. Therefore, the node construction and tree sizes are reduced, resulting in a reduction in memory consumption.

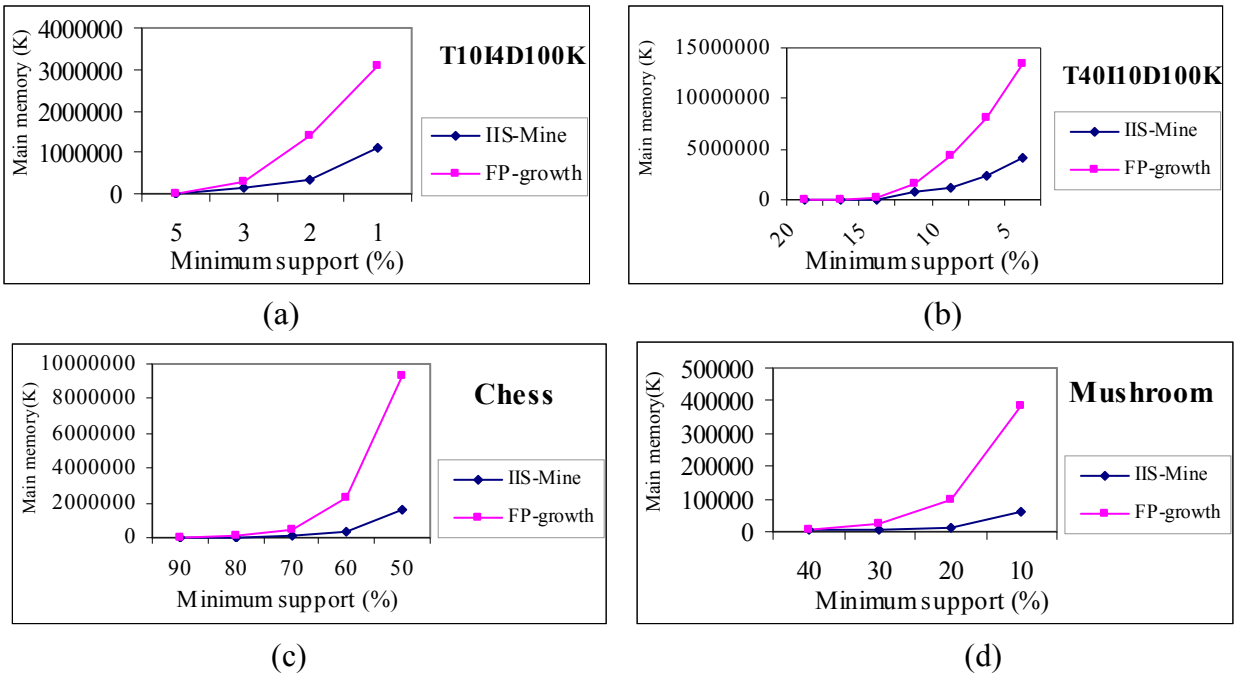


Figure 9. Memory consumption of mining on: a) T10I4D100K; b) T40I10D100K; c) Chess; d) Mushroom

Figures 9(c) and 9(d) show that the memory consumption of IIS-Mine is less than that of FP-growth on dense datasets. Figure 9(c) shows that when the minimum support is less than 80%, the memory consumption of FP-growth increases significantly compared with that of IIS-Mine. Figure 9(d) also shows that when the minimum support is less than 30%, the gap of the graphs clearly widens, which confirms that FP-growth consumes more memory than IIS-Mine. In both figures, FP-growth consumes a great deal more memory when the minimum support is low because FP-growth has constructed large FP-trees for mining all frequent itemsets, whereas IIS-Mine uses the property of extendable itemsets, which perform better for dense datasets. Consequently, the recursion of

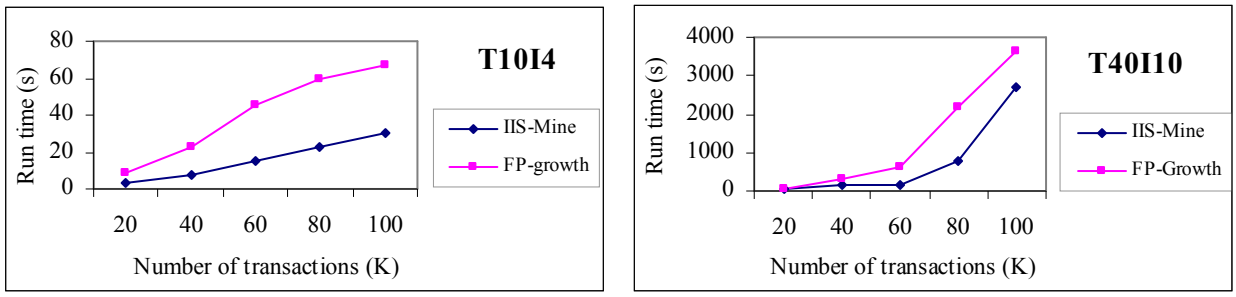
Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

mining frequent itemsets in the next loops is reduced. Therefore, the construction of nodes and the sizes of the conditional item-trees are reduced.

Scalability

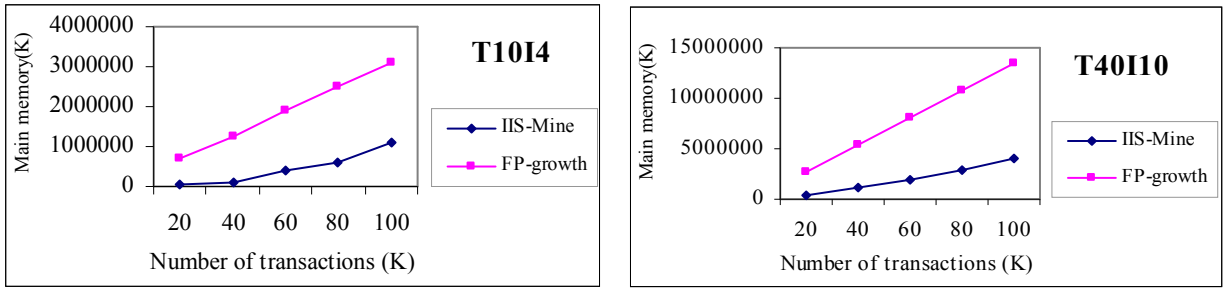
The scalability of the algorithms was tested by running them on datasets generated from T10I4 and T40I10. The number of transactions in the datasets ranged from 20*K* to 100*K*, where *K* is 1000 transactions. In Figure 10(a) and Figure 11(a), the algorithms were run on all of the datasets generated from T10I4 at a minimum support of 1%. Both run time and memory consumption were recorded. Figure 10(a) shows the speed scalability, which means that the number of transactions increases as the run time increases. Figure 11(a) shows the memory scalability of the algorithms; the curve of FP-growth is over that of IIS-Mine, which means that FP-growth consumes more memory than IIS-Mine. The figure also shows that the memory consumption of the algorithms increases linearly with the size of the datasets.

In Figures 10(b) and 11(b), the algorithms were run on all of the datasets generated from T40I10 at a minimum support of 5%. Both run time and memory consumption were recorded. Figure 10(b) confirms that the run time of both algorithms relies on the length and number of transactions: if the length or number of transactions increases, so does the run time of both algorithms. However, the run time of IIS-Mine was better than that of FP-growth for every number of transactions. Figure 11(b) confirms that the memory consumption of the algorithms increases with the length and number of transactions. However, the memory consumption of IIS-Mine was better than that of FP-growth for every number of transactions.



(a) (b)

Figure 10. Scalability of runtime on a) T10I4; b) T40I10



(a) (b)

Figure 11. Scalability of memory consumption on: a) T10I4; b) T40I10

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

CONCLUSIONS

A data structure called inverted index structure (IIS) can store transaction data by scanning a database only once. Changing the minimum support does not affect the IIS and rescanning of the database is not required. A new algorithm called IIS-Mine can find frequent itemsets without generating candidate itemsets. It employs a more efficient use of the extendable-itemset property to reduce the number of recursive steps of mining. The node construction and the size of trees are then reduced, thereby reducing the run time and memory consumption. Although the proposed method accesses the IIS structure multiple times, experimental results demonstrated that for dense datasets the IIS-Mine algorithm is better than FP-growth algorithm in run time and space consumption.

ACKNOWLEDGEMENTS

This work was supported by the National Centre of Excellence in Mathematics, PERDO, Office of the Higher Education Commission, Thailand.

REFERENCES

1. J. Han and M. Kamber, "Data Mining: Concepts and Techniques", Elsevier, Maryland Heights (MO), **2006**, pp.227-231.
2. J. Han, J. Pei, Y. Yin and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach", *Data Mining Knowl. Discov.*, **2004**, *8*, 53-87.
3. G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-Trees", *IEEE Trans. Knowl. Data Eng.*, **2005**, *17*, 1347-1362.
4. R. Agrawal, T. Imielinski and A. Swami, "Mining association rules between sets of items in large databases", Proceedings of ACM SIGMOD International Conference on Management of Data, **1993**, Washington, DC, USA, pp.207-216.
5. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", Proceedings of 20th International Conference on Very Large Data Bases, **1994**, Santiago de Chile, Chile, pp.487-499.
6. T. H. N. Vu, J. W. Lee and K. H. Ryu, "Spatiotemporal pattern mining technique for location-based service system", *ETRI J.*, **2008**, *30*, 421-431.
7. J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation", Proceedings of ACM SIGMOD International Conference on Management of Data, **2000**, Dallas (TX), USA, pp.1-12.
8. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang and D. Yang, "H-mine: Hyper-structure mining of frequent patterns in large databases", Proceedings of IEEE International Conference on Data Mining, **2001**, San Jose (CA), USA, pp.441-448.
9. A. Pietracaprina and D. Zandolin, "Mining frequent itemsets using Patricia tries", Proceedings of 3rd IEEE International Conference on Data Mining, **2003**, Melbourne (FL), USA.
10. G. Grahne and J. Zhu, "Efficiently using Prefix-Trees in mining frequent itemsets", Proceedings of 3rd IEEE International Conference on Data Mining, **2003**, Melbourne (FL), USA.

11. Q. Zhu and X. Lin, "Depth first generation of frequent patterns without candidate generation", Proceedings of 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining, **2007**, Nanjing, China, pp.378-388.
12. S. Sahaphong and V. Boonjing, "The combination approach to frequent itemsets mining", Proceedings of 3rd International Conference on Convergence and Hybrid Information Technology, **2008**, Busan, Korea, pp.565-570.
13. V. K. Shrivastava, P. Kumar and K. R. Padasani, "FP-tree and COFI based approach for mining of multiple level association rules in large database", *Int. J. Comp. Sci. Inf. Secur.*, **2010**, *7*, 273-279.
14. L. Huang, J. Z. Liang, Y. Pan and Y. Xian, "A complete attribute reduction algorithm based on improved FP tree", Proceedings of International Conference on Circuits, Communications and System, **2010**, Beijing, China, pp.1-4.
15. U. Yun and K. H. Ryu, "Approximate weighted frequent pattern mining with/without noisy environments", *Knowl.-Based Syst.*, **2011**, *24*, 73-82.
16. M. J. Zaki, "Scalable algorithms for association mining", *IEEE Trans. Knowl. Data Eng.*, **2000**, *12*, 372-390.
17. M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets", Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, **2003**, Washington, DC, USA, pp.236-355.
18. D. J. Chai, L. Jin, B. Hwang and K. H. Ryu, "Frequent pattern mining using bipartite graph", Proceedings of 18th International Conference on Database and Expert Systems Applications, **2007**, Regensburg, Germany, pp.182-186.
19. J. Dong and M. Han, "BitTableFI: An efficient mining frequent itemsets algorithm", *Knowl.-Based Syst.*, **2007**, *20*, 329-335.
20. W. Yen, "A new mining algorithm based on frequent item sets", Proceedings of International Workshop on Knowledge Discovery and Data Mining, **2008**, Adelaide, Australia, pp.410-413.
21. W. Song, B. Yang and Z. Xu, "Index-BitTableFI: An improved algorithm for mining frequent itemsets", *Knowl.-Based Syst.*, **2008**, *21*, 507-513.
22. S. Sahaphong and V. Boonjing, "Mining of frequent itemsets by using the property of extendable-itemset", Proceedings of 7th International Joint Conference on Computer Science and Software Engineering, **2010**, Bangkok, Thailand, pp.168-173.
23. S. Sahaphong and G. Sritanratana, "Mining of frequent itemsets with JoinFI-Mine algorithm", Proceedings of 10th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Database, **2011**, Cambridge, United Kingdom, pp.73-78.
24. Frequent Itemset Mining Dataset Repository, "T10I4D100K", http://fimi.cs.helsinki.fi/data/, **2003** (Accessed: January 11, 2010).
25. Frequent Itemset Mining Dataset Repository, "T40I10D100K", http://fimi.cs.helsinki.fi/data/, **2003** (Accessed: January 11, 2010).
26. Workshop on Frequent Itemset Mining Implementations, http://fimi.ua.ac.be/fimi03/, **2003** (Accessed: January 2, 2010).

Maejo Int. J. Sci. Technol. **2012**, *6*(01), 130-151

27. Workshop on Frequent Itemset Mining Implementations, http://fimi.ua.ac.be/fimi04/, **2004** (Accessed: January 2, 2010).
28. UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/, **2007** (Accessed: March 15, 2010).
29. UCI Machine Learning Repository, "Chess", http://archive.ics.uci.edu/ml/datasets.html, **1989** (Accessed: July 19, 2010).
30. UCI Machine Learning Repository, "Mushroom", http://archive.ics.uci.edu/ml/datasets.html, **1987** (Accessed: July 19, 2010).