Abstract
The aim of this paper is to develop a new mining algorithm to mine all frequent itemsets from a transaction database called the vertical index list (VIL) tree algorithm. The main advantages of the previous algorithms, which are frequent pattern (FP) growth and inverted index structure (IIS) mine, are still useful in a new approach as database scanning only done once, and all frequent itemsets are mined without generating candidate itemsets and the changing in the minimal support threshold is not affected the data structure. IIS - Mine was proposed to reduce the recursive mining steps, nodes construction, and the size of the trees. However, IIS - mine has some drawbacks when the small transaction sets are contributed to early trees, so sub - trees are of the result. To overcome this problem, VIL - Tree has been proposed to mine large transaction sets to get the early long size of frequent itemsets. This is useful when many subsets of frequent itemsets are found, and from it recursive mining steps, nodes, and sub - trees are reduced. The performance of VIL - Tree has been tested with reference to FP - growth and IIS - Mine. The experimental results demonstrate that VIL - Tree provides better performance than the two comparative algorithms in terms of run time and space consumption.

Keywords: Association rule Mining, Data mining, Frequent itemsets mining, Frequent patterns mining, Knowledge discovery

Introduction and Relate Work
Nowadays, data and information flows into organization continuously and from many sources. The abundance of data and information is collected and stored in a database, so database is always growing exponentially. It is a tremendous amount of data. The organizations want the methods to analysis data to get knowledge which hidden in large database. The knowledge is useful for organization plans such as strategic, tactical, and operational planning. The association rule mining performs to uncover interesting knowledge from huge database. It is divided into two major steps: mining of frequent itemsets and rule generation [1]. This research is performed on the first step. The innovative tactic of mining frequent itemsets separates into three categories of database scanning such as the scanning of database multiple times, two times, and one time. The algorithms in scanning a database multi times uses a level - wise algorithm and employs generate - and - test strategy for finding frequent itemsets. The total number of database scans is based on the maximum size of the frequent itemsets. The algorithm generally accepted in this group is the Apriori algorithm [2]. There are many algorithms which are good grounding in the Apriori. The AprioriTid algorithm [3] reduces the time needed for support counting procedures by replacing every transaction in the database by the set of candidate itemsets that occur in that transaction. The AprioriHybrid algorithm [4] combines the Apriori and the AprioriTid into single hybrid which has excellent scale up properties with respect to the transaction size and the number of items in the database. The DHP algorithm [5] improves the performance bottleneck of the whole process, and employs effective pruning techniques to progressively reduce the transaction database size. A sample - based frequent itemset generation algorithm was proposed in [6] that performs single pass over the data but it produces more candidate itemsets than necessary. The DIC algorithm [7] generates less candidate itemsets than the simple - based algorithm. Other algorithms can be found in the survey article at [8 - 9]. The scanning of database multi times of the algorithms in this group need to generate candidate sets which increase the I/O load and the consumption of a lot of CPU time [11]. The search space of mining explored is exponentially large. These problems have been improved by getting rid of the generate - and - test strategy and reducing the scanning of the database two times. The most popular algorithm is the FP - growth [12] which finds frequent itemsets directly from a compact tree structure without generated candidate itemsets. There are many approaches that have been proposed to extend and improve this algorithm. Jian Pei et al. developed H - mine algorithm to dynamically adjust links in the mining process [13]. The FPgrowt* algorithm reduced the FP - tree traversal time by using array technique [14]. Sahaphong and Boonjing [15] have proposed the SFI - mine algorithm which mines frequent itemsets with a new combination method without recursive construction of a conditional FP - tree. The algorithms based on the FP - growth only have to scan database twice. They mine frequent itemsets despite generating candidate itemsets. They also generate a huge conditional FP - tree which takes a lot of time and space. The vertical data layout is proposed to reduce the scanning of a given database one time. The Eclat algorithm was proposed [16] to generate all frequent itemsets in a depth - first search
and used the join step from the Apriori property, since no candidate items can be found. The data structure called large item bipartite graph was proposed to collect data [17]. Sahaphong and Boonjing [18] proposed a new algorithm that reduces run time but it generates many repeated nodes. As a result, it consumes a large amount of memory. Sahaphong and Sritanratana [19] proposed a new algorithm that uses a sorted - list structure to accommodate data when a database is scanned. Sahaphong and Boonjing [20] proposed a new approach that used an inverted index structure (IIS) and a new algorithm called IIS - Mine. The IIS - mine was proposed to reduce the recursive of mining steps, nodes construction, and the size of a given tree. However, the IIS - mine has some drawbacks. When there are small transaction sets contributing to the initial tree, so many sub - trees are still large produced. In this paper we present a new algorithm called VIL - Tree to solve the problem of IIS - Mine. The VIL - Tree uses the same basis as IIS - Mine algorithm but can reduce sub - tree by mining frequent itemsets from long transactions to get a long size in early frequent itemsets. This is to affect many subsets in the frequent itemsets that are found, so the recursive aspect of mining steps, nodes, and sub - trees are reduced. The performance of a new algorithm was tested in reference to FP - growth and IIS - mine in term of run time and space consumption. The simulation experiments used synthetics datasets. The experiment shows that VIL - Tree is more efficient than FP - growth and IIS - Mine.

Methods

Datasets

Two benchmark datasets generated by the IBM Almaden Quest research group [21] were used. There are sparse datasets which served as the FIMI repository, named T10H4D100K and T4010D100K. The notation TnlyDzK denotes a dataset where T is an average number of items per transaction, n is an average length of a frequent itemset, D is the number of transactions, and K is 1,000 transactions. Two benchmark datasets were generated by the IBM Almaden Quest research.

Tools

The experiments were performed on a notebook computer which has the following specification: the CPU being Intel ® Core ™ I5 - 2467 M (Intel ® Core™ i5 - 2467M), 1.6 GHz of clock signal, RAM is 4 GB and the operating system being Windows 7 Home Premium. All algorithms were coded using C language.

The Approach

The basic aspects of frequent itemsets mining problem are presented in [2 and 22] and all definitions are in [20]. In this paper we have presented new definitions and new algorithms to demonstrate how to mine frequent itemsets.

Definition 1. allows DB to be a transaction database. A vertical index list (VIL) is the data structure which is constructed from scanning DB once to contain transactions data. Each row in VIL consists of three attributes:

1.) An item is written in the order of descending of support items,
2.) A support of items, and
3.) The set of transactions which correspond with items in the same row and they are written in the order of their ascending identification number.

The VIL is a data structure for proposing mining algorithms. It can to apply to all minimum support thresholds without rescanning the DB. We demonstrated the steps need to construct the VIL in algorithm 1.

Algorithm 1: The instruction of VIL
Input: DB, minsup
Output: sorted VIL
Begin
Scan DB once.
Create all \( T_i \) to VIL // \( x_i \) is the items
Define all support of \( x_i \) in VIL is zero
For each \( T_i \) in DB do
For each \( x_i \) in \( T_i \) do
Insert \( T_i \) of \( x_i \) to TID-set which corresponding with \( x_i \)
Count support of \( x_i \)
End //For
End //Begin
Sort support of \( x_i \) of VIL //descending sort
End //Begin

Definition 2. allows \( T_j \) to be a transaction derived from VIL, \( R_j \) to be a set of \( T_j \) derived from each row \( j \) in VIL, where \( j = 1, 2, 3, \ldots \). Suppose that \( TR \) is the remaining transaction derived from \( TR_0 = R_1 \) and \( TR_k = R_k \cup TC_k \) where \( TC_k = R_{k+1} \setminus TR_{k+1} \) for \( k = 1, 2, \ldots, n \). According to \( FS_m(Ax) \) in Definition 8 [20], we have the following definitions.

Definition 3. allows \( A_m = \{ FS_m(Ax) \} \) is frequent itemset of length in derived from \( T \) where \( m = 2, 3, \ldots \). Let \( FS_2 = A_2 \cup \{ Ax | Ax \ is \ frequent \ itemset \ of \ length \ 2 \} \) and \( FS_m = Am \) for all \( m \geq 3 \). Suppose that \( FIm \) is derived from \( FI_1 \) \( \Rightarrow \{ x \} \) \( \subseteq \) \( Ax \) is frequent itemset and \( FIm = FS_m \) for all \( m = 2, 3, \ldots \). Any element of \( FIm \) is called frequent itemset of length of \( m \).

Definition 4. Allow \( P(Ax) \) to be a power set of \( Ax \), and any set of \( \bigcup(Ax) \) is called an extendable frequent itemset, to denote \( P_m = \{ \bigcup(P_m | Ax \ is \ frequent \ itemset) - \{ x \} \cap \bigcup(B \mid B \ is \ frequent \ itemsets \ length \ m) \} \) for all \( m \geq 2 \).
Definition 5. allows \( FI_m \) to be defined in definition 3. Suppose that \( FI \) is denoted by \( \bigcup_{m=1}^{k} FI_m \), where \( k = \max\{m|Ax \text{ would then be a frequent itemset of length } m \} \). Any element of \( FI \) is called frequent itemset.

On the basis of the definitions stated before all definitions in [20], the algorithm 2 presented the VIL - tree algorithm to show it can be used to mine all frequent itemsets.

Algorithm2: Mining frequent itemsets from VIL
Input: VIL, VIL-tree (Frequent item), and \( \text{minsup} \)
Output: \( FI \)
Procedure MiningFI (VIL, FI)
Begin
For each frequent item \( x \) in the VIL do
  // According to algorithm1
  \( FI_1 = \{ x | x \in I, \supp(x) \geq \text{minsup} \} \)
End //For
For each \( x \) in \( FI_1 \) do
  Find \( TR \)
  // \( TR \) is given in definition 2
  If \( TR \neq \phi \) then construct Tree(\( x \))
    If Tree \( \neq \phi \) then
      Call Mining-Algo (Tree, \( x \))
End //If
End //If
End //For
Find \( FI = \bigcup_{m=1}^{k} FI_m \)
  // \( FI_m \) is given in definition 3
  // \( FI \) is given in definition 5
End //Begin
Procedure Mining-Algo(Tree, \( x \))
Begin
Call SubHeader (\( A \)-tree, subheaderA)
For \( i = 1 \) to \( m \)
  Generate all \( Ax \) with its support
    // All \( Ax \) are the frequent itemsets
  // where \( A \) is the root of \( A \)-tree and
  // \( x \in \text{subheaderA} \)
For all \( Ax \) do
  If \( Ax_i \notin \text{FS}_m^* \) then
    // \( \text{FS}_m^* \) is given in [20]
    If \( \supp(Ax_i) \geq \text{minsup} \) then
      Call CondItemsetTree(\( A \)-tree, conditional \( \delta \)-tree)
      If conditional \( \delta \)-tree contains greater than two nodes then
        Call Mining-Algo (conditional \( \delta \)-tree, \( x \))
      Else Save \( \text{FS}_m^*(\delta) \) to \( FI_m \)
      If \( |\text{FS}_m^*(\delta)| \geq 2 \) then
        Find \( P_m \)
        // \( P_m \) is given in definition 4
End //Begin
Procedure SubHeader (\( A \)-tree, subheaderA)
Begin
  Each frequent item \( x \) of \( A \)-tree
    // \( \text{SH}(A) \) is given in [20]
    Find \( \{ (x,s(x)) | x \in \text{SH}(A), s(x) \text{ is the support of } x \} \)
    In \( A \)-tree
End //Begin
Procedure CondItemsetTree(Tree, \( \delta \), conditional \( \delta \)-tree)
Begin
  Scan tree once to collect the paths that have-
    An association with root \( \delta \)
  For all paths are derived from tree do
    Connect all paths to \( \delta \)
End //For
End //Begin

Result and Discussion
Runtime
Figure 1(a) and 1(b) show the experimental results of run time of the algorithms on two synthetic datasets: T10I4D100K and T40I10D100K. According to the experimental results of run time of two previous algorithms, which are FP - growth and IIS - Mine presented in [20], IIS - Mine always performs better than FP - growth. When comparing FP - growth and IIS - Mine with VIL - Tree, the experimental results of these algorithms are in the same direction. But VIL - tree is faster than the two previous algorithms in every minimal support threshold. The recursive steps of mining are still large in IIS - mine as in the case of the small transaction sets that contribute to early trees, so many sub - trees are still produced. On the contrary, VIL - tree always starts to mine frequent item which have the biggest number of transactions and provides the longest size of frequent itemsets. The number of subtrees and the size of subtrees are reduced, resulting in a reduction of run time. This is because the construction of the data structure by doing algorithm 1 and using the properties in definition 2 and definition 4.

Memory Consumption
Figure 2(a) and 2(b) show the experimental results of memory consumption of VIL - tree with two comparison algorithms which are FP - growth and IIS - Mine by using two synthetic datasets: T10I4D100K and T40I10D100K. According to experimental results of memory consumption presented in [20], FP - growth
consumes more memory than IIS - Mine in every minimal support threshold. The graphs show that the memory consumption of VIL - tree is less than the other two algorithms. This is because the construction number of nodes and trees are reduced due to a reduction of production of the new root node from the remaining transaction and also reduction is observed in repeated checks of transaction sets which are present in definition 2 and definition 4.

**Figure 1:** Run time of mining on: a) T10I4D100K; b) T40I10D100K

**Figure 2:** Memory consumption of mining on: a) T10I4D100K; b) T40I10D100K

**Conclusion**

A new algorithm called VIL - Tree mines all frequent itemsets. With this new algorithm, the performance of two comparative algorithms, FP - growth and IIS - Mine, are still useful in a proposed algorithm which are scanning database once, without generating candidate itemsets, and changing in the minimum support threshold is not affected the data structure. VIL - Tree can solve some drawbacks of
IIS - Mine, in case of the early trees are constructed to mine frequent itemsets from the small sets of transactions, this will result in reducing the performance of IIS - Mine. On the contrary, VIL - Tree creates early trees from large transaction sets to get the long sizes of frequent itemsets by using the properties in which definitions go through algorithm, so the number of the follow trees will be reduced. This improves run time and space consumption are reduced. The experiments demonstrated that VIL - Tree performs better than FP - growth and IIS - Mine in run time and space consumption.

Acknowledgements
This research was supported by the Research and Development Institute, Ramkhamhaeng University, in Bangkok Thailand. I appreciate Associate Professor Dr. Veera Boonjing for his valuable advice and Associate Professor Dr. Tawesak Tanwandee, M.D. for editing this paper.

References
