Journal of Southeast University (English Edition) Vol.19 No.4 Dec. 2003 ISSN 1003—7985

Fast FP-Growth for association rule mining

Yang Ming[1,2] Yang Ping[3] Ji Genlin[2] Sun Zhihui[2]

([1] Department of Computer Science and Engineering, Auhui University of Technology and Science, Wuhu 241000, China)

([2] Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

([3] Department of Mathematics and Physics, Anhui University of Technology and Science, Wuhu 241000, China)

Abstract: In this paper, we propose an efficient algorithm, called FFP-Growth (short for fast FP-Growth), to mine frequent itemsets. Similar to FP-Growth, FFP-Growth searches the FP-tree in the bottom-up order, but need not construct conditional pattern bases and sub-FP-trees, thus, saving a substantial amount of time and space, and the FP-tree created by it is much smaller than that created by TD-FP-Growth, hence improving efficiency. At the same time, FFP-Growth can be easily extended for reducing the search space as TD-FP-Growth (M) and TD-FP-Growth (C). Experimental results show that the algorithm of this paper is effective and efficient.

Key words: data mining; frequent itemsets; association rules; frequent pattern tree (FP-tree)

Association rule mining has many important applications in real life[1-9]. Let $I = \{i_1, i_2, i_3, \cdots, i_m\}$ be a set of binary attributes, called items. An association rule consists in an implication of the form $A => B$, where $A \subset I$, $B \subset I$ and $A \cap B = \varnothing$, i.e. A is a set of some items in I, and B is a set of some items in I that are not presented in A. The probability that B occurs given that A has occurred is called the support, and written as count (AB). The probability that B occurs given that A has occurred is called the confidence. The association rule mining problem is to find all association rules above the user-specified minimum support and minimum confidence. This is done in two steps: ① Find all frequent itemsets; ② Generate association rules from frequent itemsets where ① is the key step[2].

TD-FP-Growth[9] explores the FP-tree in the top-down order for mining frequent itemsets, and unlike Ref.[3], it need not construct conditional pattern bases and sub-trees, hence it improves efficiency. At the same time, Wang, et al.[9] extended TD-FP-Growth to mine association rules by applying two new pruning strategies: pushing multiple minimum supports — TD-FP-Growth(M) and pushing the minimum confidence — TD-FP-Growth(C). Pushing the confidence constraint into the first step can further reduce search space. But the FP-tree created by TD-FP-Growth is lexicographically ordered, so it is much larger than that created by FP-Growth. In the worst case, any two different transactions are in different paths of the FP-tree created by TD-FP-Growth, hence the size of the FP-tree is almost the same as the size of the database.

To overcome the shortcomings of TD-FP-Growth, in this paper, we propose a fast FP-Growth algorithm, called FFP-Growth, for mining frequent itemsets and association rules. FFP-Growth improves the structure of FP-tree (see definition 1). FFP-Growth also explores the FP-tree from the bottom-up, but unlike Ref.[3] it need not construct conditional patterns and sub-trees, thus, saving a substantial amount of time and space. The FP-tree created by FFP-Growth is also much smaller than that created by TD-FP-Growth. At the same time, FFP-Growth can also efficiently reduce the search space by applying pruning strategies as in Ref.[9]: pushing multiple minimum supports and pushing the minimum confidence. Experimental results show that the algorithm presented in this paper is highly effective and efficient.

1 Related Work

Since its introduction[1], the problem of mining association rules has been the subject of many studies[2-8]. But conventional Aprior-like algorithms need to repeatedly scan the database and generate lots of candidate itemsets[2,8]. To avoid generating many candidate itemsets, a frequent pattern tree (called the FP-tree) was proposed[3]. The FP-tree is searched recursively in a bottom-up order to grow longer itemsets from shorter ones. The algorithm of mining frequent itemsets[3] needs to build conditional pattern bases and

Received 2003-03-31.

Foundation items: The National Natural Science Foundation of China (79970092) and the Natural Science Foundation of Anhui Province of China (03042205).

Biography: Yang Ming (1964—), male, graduate, professor, yangming@seu.edu.cn.

sub-FP-trees for each shorter itemset in order to search for longer itemsets, thus, it becomes very time-and-space consuming as the recursion goes deep and the number of itemsets grows large.

TD-FP-Growth[9] explores the FP-tree in top-down order, and unlike Ref.[3] it need not construct conditional pattern bases and sub-FP-trees, hence it improves efficiency. But it makes the FP-tree become much larger. Unlike Ref.[9], the algorithm of this paper explores the FP-tree from the bottom-up and overcomes the drawbacks of Ref.[9] and uses less space.

Unlike Refs.[7,8], our specification of minimum support is also associated with the consequent of a rule as in Ref.[9], not with an arbitrary item or itemsets.

2 FFP-Growth for Frequent Itemsets Mining

To illustrate the idea of TD-FP-Growth, we utilize the following example[9].

Example A transaction database is given as Fig.1. Suppose that the minimum support is 2, that is, minsup = 2. After two scans of transaction database, the FP-tree (TD-FP-Growth) and the header table H are built as Fig.1. In which, an I _ node refers to a node labeled by item I. For each item I, all I _ nodes are linked by a side-link. Associated with each node v is a count, denoted by count(v), representing the number of transactions that pass through the node.

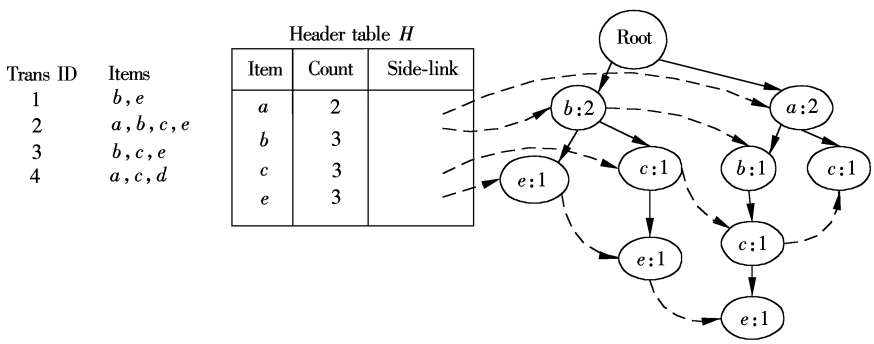

Fig.1 Transaction table, FP-tree (TD-FP-Growth) and H

The items in each path of FP-tree (TD-FP-Growth) are lexicographically ordered. According to TD-FP-Growth, frequent itemsets are found: $\{a\}$ for entry a, and $\{a, b\}$ for entry b, and $\{c\}$, $\{b, c\}$ and $\{a, c\}$ for entry c, and $\{e\}$, $\{b, e\}$, $\{c, e\}$ and $\{b, c, e\}$ for entry e, the details can be seen in Ref.[9].

Clearly, the drawbacks of TD-FP-Growth are as follows: the FP-tree (TD-FP-Growth) becomes larger, in the worst case, the size of the FP-tree (TD-FP-Growth) is almost the same as that of the database. For any entry I, it needs to search all paths that come from the root. Therefore, the structure of the FP-tree is improved as definition 1, and a fast FP-Growth algorithm is proposed, called FFP-Growth. The FP-tree created by FFP-Growth is denoted as FP-tree (FP-Growth).

Definition 1 FP-tree (FFP-Growth) is a tree structure defined below.

1) It consists of one root labeled as "null", a set of item prefix subtrees as the children of the root, and a frequent-item header table.

2) Each node in the item prefix subtree consists of four fields: item-name, count, parent, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, parent points to its parent node, and node-link links to the next node in the FP-tree (FFP-Growth) carrying the same item-name, or null if there is none.

3) Each entry in the frequent-item header table consists of three fields: ① Item-name; ② Side-link, which points to the first node in the FP-tree (FFP-Growth) carrying the item-name; and ③ Count, which registers the frequency of the item represented by the item-name in the transaction database.

Definition 2 For a node I, a path in the FP-tree (FFP-tree) starting from the root is called an I-prefixed path. Similarly, the path that comes from any itemsets α is called α-prefixed path.

As in Ref.[3], to the given example, Fig.2 is the created FP-tree (FFP-Growth). Clearly, the size of the FP-tree (FFP-Growth) is much smaller than that of the FP-tree (TD-FP-Growth).

Similar to FP-Growth, we can get all frequent itemsets, but need not construct conditional pattern bases and sub-FP-trees. For example, according to the side-link, node-link and parent pointer, we can get three paths for the last item e, $\{b\} \rightarrow \{e\}$, $\{b\} \rightarrow \{c\} \rightarrow \{e\}$ and $\{a\} \rightarrow \{b\} \rightarrow \{c\} \rightarrow \{e\}$, and quickly

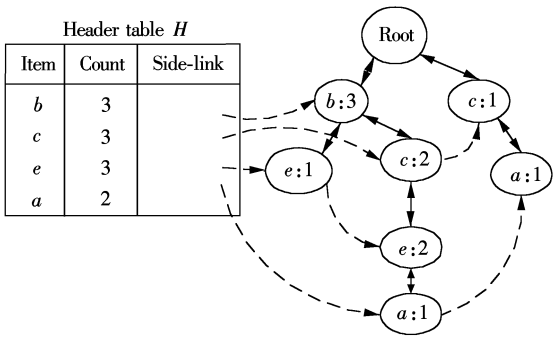

Fig.2 FP-tree (FFP-Growth) and H

get frequent itemsets $\{e\}$, $\{b, e\}$, $\{c, e\}$, $\{b, c, e\}$. Clearly, the algorithm of this paper only searches a few paths for mining frequent itemsets for the given last item. FFP-Growth is described as follows.

Algorithm FFP-Growth (fast FP-Growth)

Input: A transaction database DB and minimum support threshold minsup.

Output: The complete set of frequent itemsets F.

Methods: According to the following steps.

1) Create FP-tree (FFP-Growth);// as in Ref.[3];

2) $F = \varnothing$;

3) For each item b in the head table of FP-tree (FFP-Growth) do

{$F = F \cup \{\{b\}\}$;

Gen_FI (FP-tree (FFP-Growth), $\{b\}$, F);}

// The generated frequent itemsets are the same

// as those by FP-tree|$\{b\}$ as in Ref.[3];

4) Return F.

Procedure Gen_FI (tree, α, F)

{Find all α-prefixed paths by side-link, node-link and parent pointer in the FP-tree (FFP-Growth);

Generate all frequent items i_1, i_2, $\cdots$, i_s in all α-prefixed paths, denoted as F_1;

for $j = 1$ to s do if $\{\alpha \cup \{i_j\}\} \notin F$ then $F = F \cup \{\alpha \cup \{i_j\}\}$;

$k = 2$; CF_k = gen_candidate(F_1);// candidate itemsets CF_k

if$\{\alpha \cup \{i_1, i_2, \cdots, i_s\}\} \notin F$ then

while $CF_k \neq \varnothing$ do // frequent itemsets F_k

{ $F_k = \varnothing$;

for each A in CF_k do

{ $X = A \cup \alpha$;

if $\{X\} \notin F$ then

{get_count (FP-tree (FFP-Growth), X);

if count(X) $\geqslant$ minsup then $\{F = F \cup \{X\}$; $F_k = F_k \cup \{A\}$;}

} else $F_k = F_k \cup \{A\}$;

}

$k = k + 1$; CF_k = gen_candidate(F_{k-1});}}

Procedure get_count(tree, X)

//suppose $X = \{x_1, x_2, \cdots, x_q\}$ and count($\{x_1\}$) $\geqslant$ //count($\{x_2\}$) $\geqslant \cdots \geqslant$ count($\{x_q\}$). Clearly, X is contained in //every x_q-prefixed path, so the support of itemset X can be //quickly gotten by the parent pointer.

{ find the location of item x_q in head table;

along side-link and node-link, get all corresponding nodes nd_1, $\cdots$, nd_p;

along parent pointer of nd_1, $\cdots$, and nd_p, get all corresponding paths P_1, $\cdots$, P_p;

for ($i = 1$; $i \leqslant p$; i++)

if P_i contains X then count(X) = count(X) + nd_i.count;}

Clearly, by FFP-Growth, we need not construct conditional pattern bases and sub-FP-trees, and the FP-tree created by it is much smaller than that created by TD-FP-Growth. Moreover, to reduce the search space, FFP-Growth can be easily extended as TD-FP-Growth (M) and TD-FP-Growth (C): pushing multiple minimum supports and pushing the minimum confidence.

3 Performance Analysis

To evaluate the performance of FFP-Growth on mining frequent itemsets, we compare it with FP-Growth and TD-FP-Growth. All experiments are performed on a PIII 400 MHz Dell with 128 M main memory, running on Microsoft Windows 2000 Professional. All programs are written in VC ++ 6.0. Similar to TD-FP-Growth, we choose two datasets, Connect-4 and Forest, from UC_Irvine Machine Learning Database Repository: http://www.ics.uci.edu/~MLRepository.html (see Tab.1). The results are reported in Fig.3.

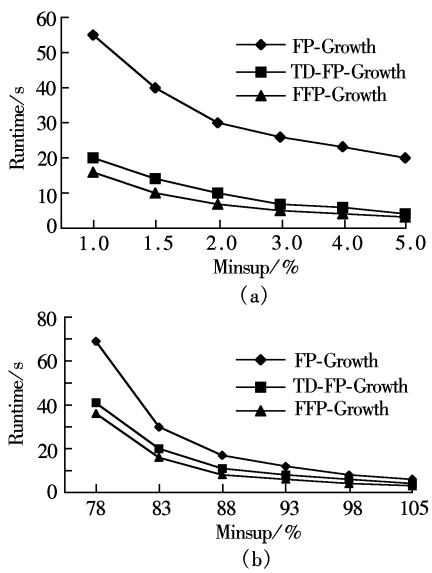

Fig.3 Executive time and scalability of algorithms. (a) Forest; (b) Connect-4

Tab.1 Data sets table

Dataset	Trans	Items per trans	Distinct items
Connect-4	67 557	43	126
Forest	581 012	13	15916

Fig.3 shows a set curves on the scalability with respect to different minimum supports (minsup). These experiments show that FFP-Growth is the most

efficient algorithm.

4 Conclusion

In this paper, we propose an efficient algorithm, called FFP-Growth. FFP-Growth need not generate conditional pattern bases and sub-FP-trees, the FP-tree created by it is much smaller than that created by TD-FP-Growth, and it only searches some paths in the FP-tree (FFP-Growth) for any given last item, hence it improves efficiency. At the same time, like TD-FP-Growth(M) and TD-FP-Growth(C), FFP-Growth can be easily extended to reduce the search space. Experimental results show that the algorithm of this paper is effective and efficient and outperforms the previous TD-FP-Growth algorithm.

References

[1] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large database [A]. In: *Proc of the ACM SIGMOD Int Conf on Management of Data* [C]. Washington DC, 1993. 207 - 216.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules [A]. In: *Proc of the 20th Int Conf Very Large Data Bases* [C]. Santiago, Chile, 1994. 487 - 499.

[3] Han J W, Pei J, Yin Y. Mining frequent patterns without candidate generation [A]. In: *Proc of the* 2000 *ACM SIGMOD Intl Conf on management of data* [C]. Dallas, 2000.1 - 12.

[4] Han J W, Pei J, Yin Y. Mining partial periodicity using frequent pattern trees [R]. Canada: Simon Fraser University, Computing Science Technical Report: TR-99-10, 1999.

[5] Srikant R, Agrawal R. Mining generalized association rules [A]. In: *Proc of the* 21*st Int Conf on Very Large DataBases* [C]. Zurich, Switzerland, 1995. 407 - 419.

[6] Srikant R, Yu Q, Agrawal R. Mining association rules with item constraints [A]. In: *Proc of the KDD* [C]. Newport Beach, CA, 1997. 67 - 73.

[7] Liu B, Hsu W, Ma Y. Mining association rules with multiple minimum supports [A]. In: *Proc of ACM SIGKDD* [C]. San Diego, CA, 1999.337 - 341.

[8] Wang K, He Y, Han J W. Mining frequent itemsets using support constraints [A]. In: *Proc Int Conf on Very Large Data Bases* [C]. Cairo, 2000. 43 - 52.

[9] Wang K, Tang L, Han J W, et al. Top down FP-Growth for association rule mining [A]. In: *Proc of the 6th Pacific Area Conf on Knowledge Discovery and Data Mining* [C]. Taipei, 2002.

基于频繁模式树的快速关联规则挖掘算法

杨 明[1,2] 杨 萍[3] 吉根林[2] 孙志挥[2]

([1] 安徽工程科技学院计算机科学与工程系,芜湖 241000)

([2] 东南大学计算机科学与工程系, 南京 210096)

([3] 安徽工程科技学院应用数理系, 芜湖 241000)

摘 要 提出了一种挖掘频繁项目集的有效算法——FFP-Growth,该算法采用自底向上的策略搜索频繁模式树,但不同于 FP-Growth 的是它无须生成条件模式基和频繁模式子树,且生成的频繁模式树较 TD-FP-Growth 生成的频繁模式树小,因而能提高关联规则的挖掘效率. 类似于 TD-FP-Growth 的扩展 TD-FP-Growth(M) 和 TD-FP-Growth(C),FFP-Growth 很容易被扩展,以此来有效地减小搜索空间. 实验结果表明本文提出的算法是有效可行的.

关键词 数据挖掘; 频繁项目集; 关联规则; 频繁模式树

中图分类号 TP311