Non Recursive Generation of Frequent K-itemsets from Frequent Pattern Tree Representations

Mohammad El-Hajj and Osmar R. Zaïane

Department of Computing Science, University of Alberta, Edmonton AB, Canada
{mohammad, zaiane}@cs.ualberta.ca

Abstract. Existing association rule mining algorithms suffer from many problems when mining massive transactional datasets. One major problem is the high memory dependency: gigantic data structures built are assumed to fit in main memory; in addition, the recursive mining process to mine these structures is also too voracious in memory resources. This paper proposes a new association rule-mining algorithm based on frequent pattern tree data structure. Our algorithm does not use much more memory over and above the memory used by the data structure. For each frequent item, a relatively small independent tree called COFI-tree, is built summarizing co-occurrences. Finally, a simple and non-recursive mining process mines the COFI-trees. Experimental studies reveal that our approach is efficient and allows the mining of larger datasets than those limited by FP-Tree

1 Introduction

Recent days have witnessed an explosive growth in generating data in all fields of science, business, medicine, military, etc. The same rate of growth in the processing power of evaluating and analyzing the data did not follow this massive growth. Due to this phenomenon, a tremendous volume of data is still kept without being studied. Data mining, a research field that tries to ease this problem, proposes some solutions for the extraction of significant and potentially useful patterns from these large collections of data. One of the canonical tasks in data mining is the discovery of association rules. Discovering association rules, considered as one of the most important tasks, has been the focus of many studies in the last few years. Many solutions have been proposed using a sequential or parallel paradigm. However, the existing algorithms depend heavily on massive computation that might cause high dependency on the memory size or repeated I/O scans for the data sets. Association rule mining algorithms currently proposed in the literature are not sufficient for extremely large datasets and new solutions, that especially are less reliant on memory size, still have to be found.

1.1 Problem Statement

The problem consists of finding associations between items or itemsets in transactional data. The data could be retail sales in the form of customer transactions

or any collection of sets of observations. Formally, as defined in [2], the problem is stated as follows: Let $I = \{i_1, i_2, ...i_m\}$ be a set of literals, called items. m is considered the dimensionality of the problem. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A unique identifier *TID* is given to each transaction. A transaction T is said to contain X, a set of items in I, if $X \subseteq T$. An *association rule* is an implication of the form "$X \Rightarrow Y$", where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$. An itemset X is said to be *large* or *frequent* if its *support* s is greater or equal than a given minimum support threshold σ. The rule $X \Rightarrow Y$ has a *support* s in the transaction set $\mathcal{D}$ if $s\%$ of the transactions in $\mathcal{D}$ contain $X \cup Y$. In other words, the support of the rule is the probability that X and Y hold together among all the possible presented cases. It is said that the rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if $c\%$ of transactions in $\mathcal{D}$ that contain X also contain Y. In other words, the confidence of the rule is the conditional probability that the consequent Y is true under the condition of the antecedent X. The problem of discovering all association rules from a set of transactions $\mathcal{D}$ consists of generating the rules that have a *support* and *confidence* greater than a given threshold. These rules are called *strong rules*. This association-mining task can be broken into two steps: 1. A step for finding all frequent k-itemsets known for its extreme I/O scan expense, and the massive computational costs; 2. A straightforward step for generating strong rules. In this paper, we are mainly interested in the first step.

1.2 Related Work

Several algorithms have been proposed in the literature to address the problem of mining association rules [2, 6]. One of the key algorithms, which seems to be the most popular in many applications for enumerating frequent itemsets is the *Apriori* algorithm [2]. This *Apriori* algorithm also forms the foundation of most known algorithms. It uses a monotone property stating that for a k-itemset to be frequent, all its (k-1)-itemsets have to be frequent. The use of this fundamental property reduces the computational cost of candidate frequent itemset generation. However, in the cases of extremely large input sets with big frequent 1-items set, the *Apriori* algorithm still suffers from two main problems of repeated I/O scanning and high computational cost. One major hurdle observed with most real datasets is the sheer size of the candidate frequent 2-itemsets and 3-itemsets. TreeProjection is an efficient algorithm presented in [1]. This algorithm builds a lexicographic tree in which each node of this tree presents a frequent pattern. The authors of this algorithm report that their algorithm is one order of magnitude faster than the existing techniques in the literature. Another innovative approach of discovering frequent patterns in transactional databases, FP-Growth, was proposed by Han et al. in [6]. This algorithm creates a compact tree-structure, FP-Tree, representing frequent patterns, that alleviates the multi-scan problem and improves the candidate itemset generation. The algorithm requires only two full I/O scans of the dataset to build the prefix tree in main memory and then mines directly this structure. The authors

of this algorithm report that their algorithm is faster than the *Apriori* and the TreeProjection algorithms. Mining the FP-tree structure is done recursively by building conditional trees that are of the same order of magnitude in number as the frequent patterns. This massive creation of conditional trees makes this algorithm not scalable to mine large datasets beyond few millions. [7] proposes a new algorithm H-mine that invokes FP-Tree to mine condensed data. This algorithm is still not scalable as reported by its authors in [8].

1.3 Preliminaries, Motivations and Contributions

The (Co-Occurrence Frequent Item Tree, or COFI-tree for short) algorithm that we are presenting in this paper is based on the core idea of the FP-Growth algorithm proposed by Han et al. in [6]. A compacted tree structure, FP-Tree, is built based on an ordered list of the frequent 1-itemsets present in the transactional database. However, rather than using FP-Growth which recursively builds a large number of relatively large trees called conditional trees [6] from the built FP-tree, we successively build one small tree (called COFI-tree) for each frequent 1-itemset and mine the trees with simple non-recursive traversals. We keep only one such COFI-tree in main memory at a time.

The COFI-tree approach is a divide and conquer approach, in which we do not seek to find all frequent patterns at once, but we independently find all frequent patterns related to each frequent item in the frequent 1-itemset. The main differences between our approach and the FP-growth approach are the followings: (1) we only build one COFI-tree for each frequent item A. This COFI-tree is non-recursively traversed to generate all frequent patterns related to item A. (2) Only one COFI-tree resides in memory at one time and it is discarded as soon as it is mined to make room for the next COFI-tree.

Algorithms like FP-Tree-based depend heavily on the memory size as the memory size plays an important role in defining the size of the problem. Memory is not only needed to store the data structure itself, but also to generate recursively in the mining process the set of conditional trees. This phenomenon is often overlooked. As argued by the authors of the algorithm, this is a serious constraint [8]. Other approaches such as in [7], build yet another data structure from which the FP-Tree is generated, thus doubling the need for main memory.

The current association rule mining algorithms handle only relatively small sizes with low dimensions. Most of them scale up to only a couple of millions of transactions and a few thousands of dimensions [8, 5]. None of the existing algorithms scales to beyond 15 million transactions, and hundreds of thousands of dimensions, in which each transaction has an average of at least a couple of dozen items.

The remainder of this paper is organized as follows: Section 2 describes the Frequent Pattern tree, design and construction. Section 3 illustrates the design, constructions and mining of the Co-Occurrence Frequent Item trees. Experimental results are given in Section 4. Finally, Section 5 concludes by discussing some issues and highlights our future work.

2 Frequent Pattern Tree: Design and Construction

The COFI-tree approach we propose consists of two main stages. Stage one is the construction of the Frequent Pattern tree and stage two is the actual mining for these data structures, much like the FP-growth algorithm.

2.1 Construction of the Frequent Pattern Tree

The goal of this stage is to build the compact data structures called Frequent Pattern Tree [6]. This construction is done in two phases, where each phase requires a full I/O scan of the dataset. A first initial scan of the database identifies the frequent 1-itemsets. The goal is to generate an ordered list of frequent items that would be used when building the tree in the second phase.

This phase starts by enumerating the items appearing in the transactions. After enumeration these items (i.e. after reading the whole dataset), infrequent items with a support less than the support threshold are weeded out and the remaining frequent items are sorted by their frequency. This list is organized in a table, called header table, where the items and their respective support are stored along with pointers to the first occurrence of the item in the frequent pattern tree. Phase 2 would construct a frequent pattern tree.

Table 1. Transactional database

T.No.	Items					T.No.	Items					T.No.	Items					T.No.	Items				
T1	A	G	D	C	B	T2	B	C	H	E	D	T3	B	D	E	A	M	T4	C	E	F	A	N
T5	A	B	N	O	P	T6	A	C	Q	R	G	T7	A	C	H	I	G	T8	L	E	F	K	B
T9	A	F	M	N	O	T10	C	F	P	G	R	T11	A	D	B	H	I	T12	D	E	B	K	L
T13	M	D	C	G	O	T14	C	F	P	Q	J	T15	B	D	E	F	I	T16	J	E	B	A	D
T17	A	K	E	F	C	T18	C	D	L	B	A												

Fig. 1. Steps of phase 1.

Phase 2 of constructing the Frequent Pattern tree structure is the actual building of this compact tree. This phase requires a second complete I/O scan

from the dataset. For each transaction read only the set of frequent items present in the header table is collected and sorted in descending order according to their frequency. These sorted transaction items are used in constructing the FP-Trees as follows: for the first item on the sorted transactional dataset, check if it exists as one of the children of the root. If it exists then increment the support for this node. Otherwise, add a new node for this item as a child for the root node with 1 as support. Then, consider the current item node as the newly temporary root and repeat the same procedure with the next item on the sorted transaction. During the process of adding any new item-node to the FP-Tree, a link is maintained between this item-node in the tree and its entry in the header table. The header table holds as one pointer per item that points to the first occurrences of this item in the FP-Tree structure.

For illustration, we use an example with the transactions shown in Table 1. Let the minimum support threshold set to 4. Phase 1 starts by accumulating the support for all items that occur in the transactions. Step 2 of phase 1 removes all non-frequent items, in our example (G, H, I, J, K, L,M, N, O, P, Q and R), leaving only the frequent items (A, B, C, D, E, and F). Finally all frequent items are sorted according to their support to generate the sorted frequent 1-itemset. This last step ends phase 1 of the COFI-tree algorithm and starts the second phase. In phase 2, the first transaction (A, G, D, C, B) read is filtered to consider only the frequent items that occur in the header table (i.e. A, D, C and B). This frequent list is sorted according to the items' supports (A, B, C and D). This ordered transaction generates the first path of the FP-Tree with all item-node support initially equal to 1. A link is established between each item-node in the tree and its corresponding item entry in the header table. The same procedure is executed for the second transaction (B, C, H, E, and D), which yields a sorted frequent item list (B, C, D, E) that forms the second path of the FP-Tree. Transaction 3 (B, D, E, A, and M) yields the sorted frequent item list (A, B, D, E) that shares the same prefix (A, B) with an existing path on the tree. Item-nodes (A and B) support is incremented by 1 making the support of (A) and (B) equal to 2 and a new sub-path is created with the remaining items on the list (D, E) all with support equal to 1. The same process occurs for all transactions until we build the FP-Tree for the transactions given in Table 1. Figure 2 shows the result of the tree building process.

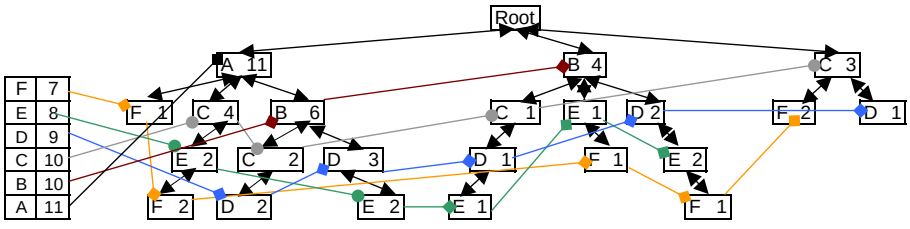

Fig. 2. Frequent Pattern Tree.

3 Co-Occurrence Frequent-Item-trees: Construction and Mining

Our approach for computing frequencies relies first on building independent relatively small trees for each frequent item in the the header table of the FP-Tree called COFI-trees. Then we mine separately each one of the trees as soon as they are built, minimizing the candidacy generation and without building conditional sub-trees recursively. The trees are discarded as soon as mined. At any given time, only one COFI-tree is present in main memory.

3.1 Construction of the Co-Occurrence Frequent-Item-trees

The small COFI-trees we build are similar to the conditional FP-trees in general in the sense that they have a header with ordered frequent items and horizontal pointers pointing to a succession of nodes containing the same frequent item, and the prefix tree per-se with paths representing sub-transactions. However, the COFI-trees have bidirectional links in the tree allowing bottom-up scanning as well, and the nodes contain not only the item label and a frequency counter, but also a participation counter as explained later in this section. The COFI-tree for a given frequent item x contains only nodes labeled with items that are more frequent or as frequent as x.

To illustrate the idea of the COFI-trees, we will explain step by step the process of creating COFI-trees for the FP-Tree of Figure 2. With our example, the first Co-Occurrence Frequent Item tree is built for item F as it is the least frequent item in the header table. In this tree for F, all frequent items which are more frequent than F and share transactions with F participate in building the tree. This can be found by following the chain of item F in the FP-Tree structure. The F-COFI-tree starts with the root node containing the item in question, F. For each sub-transaction or branch in the FP-Tree containing item F with other frequent items that are more frequent than F which are parent nodes of F, a branch is formed starting from the root node F. The support of this branch is equal to the support of the F node in its corresponding branch in FP-Tree. If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly. Figure 3 illustrates all COFI-trees for frequent items of Figure 2. In Figure 3, the rectangle nodes are nodes from the tree with an item label and two counters. The first counter is a *support-count* for that node while the second counter, called *participation-count*, is initialized to 0 and is used by the mining algorithm discussed later, a horizontal link which points to the next node that has the same *item-name* in the tree, and a bi-directional vertical link that links a child node with its parent and a parent with its child. The bi-directional pointers facilitate the mining process by making the traversal of the tree easier. The squares are actually cells from the header table as with the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains

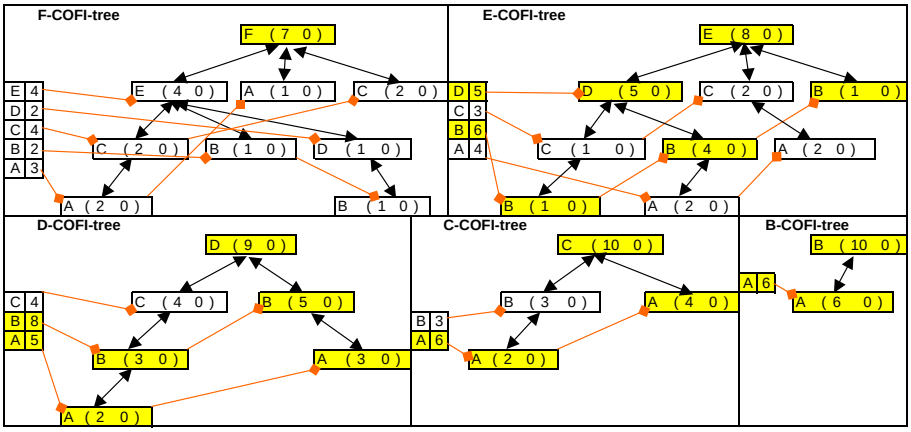

Fig. 3. COFI-trees

the *item-name*, *item-counter*, and a *pointer* to the first node in the tree that has the same *item-name*.

To explain the COFI-tree building process, we will highlight the building steps for the F-COFI-tree in Figure 3. Frequent item F is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item F indicate that it shares a branch with item A, with support = 1 for this branch as the support of the F-item is considered the support for this branch (following the upper links for this item). Two nodes are created, for FA: 1. The second location of F indicate a new branch of FECA:2 as the support of F=2. Three nodes are created for items ECA with support = 2. The support of the F node is incremented by 2. The third location indicates the sub-transaction FEB:1. Nodes for F and E are already exist and only new node for B is created as a another child for E. The support for all these nodes are incremented by 1. B becomes 1, E becomes 3 and F becomes 4. FEDB:1 is read after that, FE branch already exists and a new child branch for DB is created as a child for E with support = 1. The support for E nodes becomes 4, F becomes 5. Finally FC:2 is read, and a new node for item C is created with support =2, and F support becomes 7. Like with FP-Trees, the header constitutes a list of all frequent items to maintain the location of first entry for each item in the COFI-tree. A link is also made for each node in the tree that points to the next location of the same item in the tree if it exists. The mining process is the last step done on the F-COFI-tree before removing it and creating the next COFI-tree for the next item in the header table.

3.2 Mining the COFI-trees

The COFI-trees of all frequent items are not constructed together. Each tree is built, mined, then discarded before the next COFI-tree is built. The mining pro-

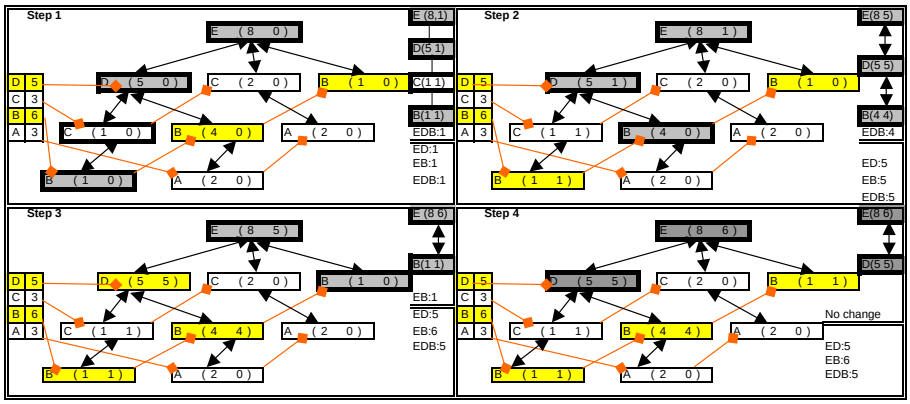

Fig. 4. Steps needed to generate frequent patterns related to item E

cess is done for each tree independently with the purpose of finding all frequent k-itemset patterns in which the item on the root of the tree participates.

Steps to produce frequent patterns related to the E item for example, are illustrated in Figure 4. From each branch of the tree, using the *support-count* and the *participation-count*, candidate frequent patterns are identified and stored temporarily in a list. The non-frequent ones are discarded at the end when all branches are processed. The mining process for the E-COFI-tree starts from the most locally frequent item in the header table of the tree, which is item B. Item B exists in three branches in the E-COFI-tree which are (B:1, C:1, D:5 and E:8), (B:4, D:5, and E:8) and (B:1, and E:8). The frequency of each branch is the frequency of the first item in the branch minus the participation value of the same node. Item B in the first branch has a frequency value of 1 and participation value of 0 which makes the first pattern EDB frequency equals to 1. The participation values for all nodes in this branch are incremented by 1, which is the frequency of this pattern. In the first pattern EDB: 1, we need to generate all sub-patterns that item E participates in which are ED:1 EB:1 and EDB:1. The second branch that has B generates the pattern EDB: 4 as the frequency of B on this branch is 4 and its participation value is equal to 0. All participation values on these nodes are incremented by 4. Sub-patterns are also generated from the EDB pattern which are ED: 4 , EB: 4, and EDB: 4. All patterns already exist with support value equals to 1, and only updating their support value is needed to make it equal to 5. The last branch EB:1 will generate only one pattern which is EB:1, and consequently its value will be updated to become 6. The second locally frequent item in this tree, "D" exists in one branch (D: 5 and E: 8) with participation value of 5 for the D node. Since the participation value for this node equals to its support value, then no patterns can be generated from this node. Finally all non-frequent patterns are omitted leaving us with only frequent patterns that item E participates in which are ED:5, EB:6 and EBD:5. The COFI-tree of Item E can be removed at this time

and another tree can be generated and tested to produce all the frequent patterns related to the root node. The same process is executed to generate the frequent patterns. The D-COFI-tree is created after the E-COFI-tree. Mining this tree generates the following frequent patterns: DB:8, DA:5, and DBA:5. C-COFI-tree generates one frequent pattern which is CA:6. Finally, the B-COFI-tree is created and the frequent pattern BA:6 is generated.

4 Experimental Evaluations and Performance Study

To test the efficiency of the COFI-tree approach, we conducted experiments comparing our approach with two well-known algorithms namely: *Apriori* and FP-Growth. To avoid implementation bias, third party *Apriori* implementation, by Christian Borgelt [4], and FP-Growth [6] written by its original authors are used. The experiments were run on a 733-MHz machine with a relatively small RAM of 256MB.

Transactions were generated using IBM synthetic data generator [3]. We conducted different experiments to test the COFI-tree algorithm when mining extremely large transactional databases. We tested the applicability and scalability of the COFI-tree algorithm. In one of these experiments, we mined using a support threshold of 0.01% transactional databases of sizes ranging from 1 million to 25 million transactions with an average transaction length of 24 items. The dimensionality of the 1 and 2 million transaction dataset was 10,000 items while the datasets ranging from 5 million to 25 million transactions had a dimensionality of 100,000 unique items. Figure 5A illustrates the comparative results obtained with *Apriori*, FP-Growth and the COFI-tree. *Apriori* failed to mine the 5 million transactional database and FP-Growth couldn't mine beyond the 5 million transaction mark. The COFI-tree, however, demonstrates good scalability as this algorithm mines 25 million transactions in 2921s (about 48 minutes). None of the tested algorithms, or reported results in the literature reaches such a big size. To test the behavior of the COFI-tree vis-à-vis different support thresholds, a set of experiments was conducted on a database size of one million transactions, with 10,000 items and an average transaction length of 24 items. The mining process tested different support levels, which are 0.0025% that revealed almost 125K frequent patterns, 0.005% that revealed nearly 70K frequent patterns, 0.0075% that generated 32K frequent patterns and 0.01 that returned 17K frequent patterns. Figure 5B depicts the time needed in seconds for each one of these runs. The results show that the COFI-tree algorithm outperforms both *Apriori* and FP-Growth algorithms in all cases.

5 Discussion and Future Work

Finding scalable algorithms for association rule mining in extremely large databases is the main goal of our research. To reach this goal, we propose a new algorithm that is FP-Tree based. This algorithm identifies the main problem of the FP-Growth algorithm which is the recursive creation and mining of many conditional

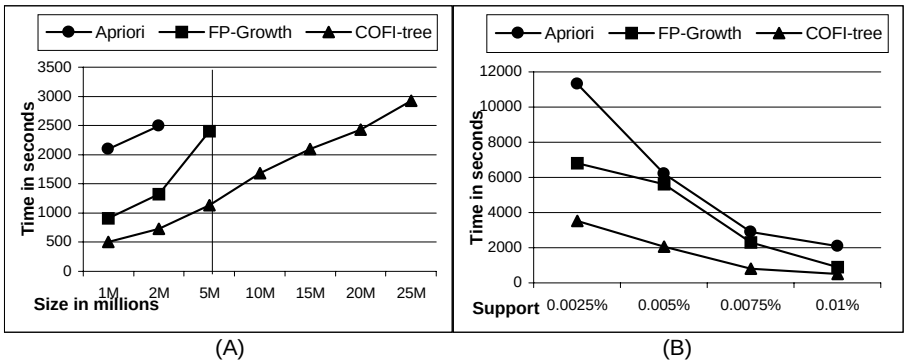

Fig. 5. Computational performance and scalability

pattern trees, which are equal in number to the distinct frequent patterns generated. We have replaced this step by creating one COFI-tree for each frequent item. A simple non-recursive mining process is applied to generate all frequent patterns related to the tested COFI-tree. The experiments we conducted showed that our algorithm is scalable to mine tens of millions of transactions, if not more. We are currently studying the possibility of parallelizing the COFI-tree algorithm to investigate the opportunity of mining hundred of millions of transactions in a reasonable time and with acceptable resources.

References

1. R. Agarwal, C.Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Parallel and distributed Computing*, 2000.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.
3. IBM. Almaden. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html.
4. C. Borgelt. Apriori implementation. http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori/apriori.html.
5. E.-H. Han, G. Karypis, and V.Kumar. Scalable parallel data mining for association rule. *Transactions on Knowledge and data engineering*, 12(3):337–352, May-June 2000.
6. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD*, Dallas, 2000.
7. H. Huang, X. Wu, and R. Relue. Association analysis with one scan of databases. In *IEEE International Conference on Data Mining*, pages 629–636, December 2002.
8. J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by oppotunistic projection. In *Eight ACM SIGKDD Internationa Conf. on Knowledge Discovery and Data Mining*, pages 229–238, Edmonton, Alberta, August 2002.