Parallel Association Rule Mining with Minimum Inter-Processor Communication[*]

Mohammad El-Hajj
Department of Computing Science
University of Alberta Edmonton, AB, Canada
mohammad@cs.ualberta.ca

Osmar R. Zaïane
Department of Computing Science
University of Alberta Edmonton, AB, Canada
zaiane@cs.ualberta.ca

Abstract

Existing parallel association rule mining algorithms suffer from many problems when mining massive transactional datasets. One major problem is that most of the parallel algorithms for a shared nothing environment are Apriori-based algorithms. Apriori-based algorithms are proven to be not scalable due to many reasons, mainly: (1) the repetitive I/O disk scans, (2) the huge computation and communication involved during the candidacy generation.

This paper proposes a new disk-based parallel association rule mining algorithm called Inverted Matrix, which achieves its efficiency by applying three new ideas. First, transactional data is converted into a new database layout called Inverted Matrix that prevents multiple scanning of the database during the mining phase, in which finding globally frequent patterns could be achieved in less than a full scan with random access. This data structure is replicated among the parallel nodes. Second, for each frequent item assigned to a parallel node, a relatively small independent tree is built summarizing co-occurrences. Finally, a simple and non-recursive mining process reduces the memory requirements as minimum candidacy generation and counting is needed, and no communication between nodes is required to generate all globally frequent patterns.

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 of the datasets. The published parallel implementations of association rule mining inherited most of these problems in addition to the new costly communication cost most of them need. Parallel association rule mining algorithms currently proposed in the literature are not sufficient for extremely large datasets, and new solutions, that do not heavily depend on repeated I/O scan, less reliant on memory size, and do not require a lot of communication costs between nodes, still have to be found.

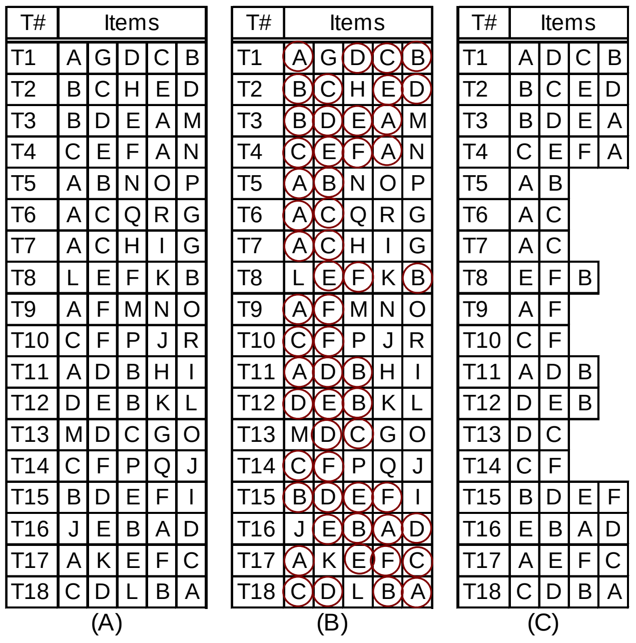
In this paper we are introducing a new parallel association rule mining disk-based algorithm that is based on the COFI-trees concept discussed in detail in this paper. This algorithm is divided into two main phases. The first one, considered pre-processing, requires only two full I/O scans of the dataset and generates a special disk-based data structure called Inverted Matrix. In the second phase, the Inverted Matrix is replicated among nodes and mined in parallel using different support levels to generate association rules using the Inverted Matrix algorithm explained later in this paper. The mining process might take, in some cases, less than one-full I/O scan of the data structure in which only frequent items based on the support given by the user are scanned and participate in generating the frequent patterns.

The remainder of this paper is organized as follows: Section 2 illustrates the transactional layout and the motivations of the Inverted Matrix approach. Section 3 describes the design and constructions of the parallel Co-Occurrence Frequent Item trees. Section 4 concludes by discussing some issues and highlights our future work.

[*]This research is partially supported by a Research Grant from NSERC, Canada.

2 Transaction Layout

Frequent itemset mining algorithms mine the database on a given fixed support threshold. If the support threshold changes, the mining process is repeated and the previous accumulated knowledge is not taken into account. For instance, in the simple transactional database of Figure 1A, where each line represents a transaction (called horizontal layout), we can observe that when changing support one can avoid reading some entries. If the support level is greater than 4, then Figure 1B highlights all frequent items that need to be scanned and computed. Non-circled items in Figure 1B are not included in the generation of the frequent items, and reading them becomes useless. It is known that all of the existing algorithms scan the whole database, frequent and non-frequent items more than once, generating a huge amount of useless work [2]. We call this superfluous processing. Figure 1C represents what we actually need to read and compute from the transactional database based on a support greater than 4. Obviously, this may not be possible with this horizontal layout, but with a vertical layout avoiding these useless reads is possible. The transaction layout is



(A)

T#	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	J	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

(B)

T#	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	J	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

(C)

T#	Items			
T1	A	D	C	B
T2	B	C	E	D
T3	B	D	E	A
T4	C	E	F	A
T5	A	B		
T6	A	C		
T7	A	C		
T8	E	F	B	
T9	A	F		
T10	C	F		
T11	A	D	B	
T12	D	E	B	
T13	D	C		
T14	C	F		
T15	B	D	E	F
T16	E	B	A	D
T17	A	E	F	C
T18	C	D	B	A

Figure 1. A: Transactional database (B): Frequent items circled (C): Needed Items to be scanned, $\sigma > 4$.

the method in which items in transactions are formatted in the database. Currently, there are two approaches: the horizontal approach and the vertical approach. In this section these approaches are discussed and a new transactional layout called Inverted Matrix is presented and compared with the existing two methods.

2.1 Horizontal vs. Vertical layout

The format of storing transactions in the database plays an important role in determining the efficiency of the association rule-mining algorithm used. Existing algorithms use one of the two layouts, namely horizontal and vertical. The first one, which is the most commonly used, relates all items on the same transaction together. In this approach the ID of the transaction plays the role of the key for the transactional table. Figure 1A represents a sample of 18 transactions made of 18 items. The vertical layout relates all transactions that share the same items together. In this approach the key of each record is the item. Each record in this approach has an item with all transaction numbers in which this item occurs. The horizontal layout has a very important advantage, which is combining all items in one transaction together. In this layout and by using some clever techniques, such as the one used in [4], the candidacy generation step can be eliminated. On the other hand, this layout suffers from limitations such as the problem mentioned above that we called superfluous processing since there is no index on the items. The vertical layout, however, is an index on the items in itself and reduces the effect of large data sizes as there is no need to always re-scan the whole database. On the other hand, this vertical layout still needs the expensive candidacy generation phase. Also computing the frequencies of itemsets becomes the tedious task of intersecting records of different items of the candidate patterns.

2.2 Inverted Matrix Layout

The Inverted Matrix layout combines the two previously mentioned layouts with the purpose of making use of the best of the two approaches and reducing their drawbacks as much as possible. The idea of this approach is to associate each item with all transactions in which it occurs (i.e. an inverted index), and to associate each transaction with all its items using pointers. Similar to the vertical layout, the item is the key of each record in this layout. The difference between this layout and the vertical layout is that each attribute on the Inverted Matrix is not the transaction ID, but a pointer that points to the location of the next item on the same transaction. The transaction ID could be preserved in our layout, but since it is not needed for the purpose of frequent itemset mining, it is discarded. The pointer is a pair where the first element indicates the address of a line in the matrix and the second element indicates the address of a column. Each line in the matrix has an address (sequential number in our illustrative example) and is prefixed by the item it represents with its frequency in the database. The lines are ordered in ascending order of the frequency of the item they represent. Figure 2 represents the Inverted Matrix corresponding to the transactional database from Figure 1A. Building this Inverted Matrix is done in two

loc	Index	Transactional Array										
		1	2	3	4	5	6	7	8	9	10	11
1	R,2	2,1	3,2									
2	Q,2	12,2	3,3									
3	P,3	4,1	9,1	9,2								
4	O,3	5,2	5,3	6,3								
5	N,3	13,1	17,4	6,2								
6	M,3	14,2	13,3	12,4								
7	L,3	8,1	8,2	15,9								
8	K,3	13,2	14,5	13,7								
9	J,3	13,4	13,5	14,7								
10	I,3	11,2	11,3	13,6								
11	H,3	14,1	12,3	15,4								
12	G,4	15,1	16,4	16,5	16,5							
13	F,7	14,3	14,4	18,7	16,6	16,8	14,6	14,8				
14	E,8	15,2	15,3	16,3	17,5	15,5	15,7	15,8	16,9			
15	D,9	16,1	16,2	17,3	17,6	17,7	16,7	17,8	17,9	16,10		
16	C,10	17,1	17,2	18,3	18,5	18,6	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	18,10	17,10	
17	B,10	18,1	ϕ,ϕ	18,2	18,4	ϕ,ϕ	18,8	ϕ,ϕ	ϕ,ϕ	18,9	18,11	
18	A,11	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ	ϕ,ϕ

Figure 2. Inverted Matrix

phases, in which phase one scans the database once to find the frequency of each item and orders them into ascending order, The second phase scans the database again once to sort each transaction into ascending order according to the frequency of each item, and then fills in the matrix appropriately. To illustrate the process, let us consider the construction of the matrix in Figure 2. The first transaction in Figure 1A has items (A, G, D, C, B). This transaction is sorted into (G, D, C, B, A) based on the item frequencies. process. Item G has the physical location line 12 in the Inverted Matrix in Figure 2, D has the location line 15, the location of C is line 16, B is in line 17 and finally A is in line 18. This is according to the vertical approach. Item G has a link to the first empty slot in the transactional array of item D that is 1. Consequently, (15,1) entry is added in the first slot of item G to point to the first empty location in the transactional array of D. At the First empty location of D (15,1) an entry is added to point to the first empty location of the next item C that is (16,1). The same process occurs for all items in the transaction. The last item of the transaction, item A produces an entry with pointer null (ϕ,ϕ). The same is performed for every transaction.

3 Parallel Co-Occurrence Frequent Item-trees: Design and Construction

The generation of frequencies is considered a costly operation for association rule discovery. In *Apriori*-based algorithms this step might become a complex problem in cases of high dimensionality due to the sheer size of the candidacy generation [2]. In methods such as FP-Growth [4], the candidacy generation is replaced by a recursive routine that builds a very large number of sub-trees, called conditional FP-trees, that are on the same order of magnitude as number of the frequent patterns.

Our approach for computing frequencies relies first on replicating the Inverted Matrix among all parallel nodes. By doing so a full set of transactional database would be available to each processor to generate all globally frequent patterns with minimum communication cost at parallel node level. This replication is executed from a designated master node. Second, this approach relies on distributing frequent items among the parallel nodes. Our strategy for evenly distributing the frequent items to the different processors such that the load is as balanced as possible, is explained later. Each parallel node is responsible for generating all frequent patterns related to the frequent items associated to that node. To do so each node reads sub-transactions for frequent items directly from the Inverted Matrix, then builds independent and relatively small trees for each frequent item in the transactional database. Each parallel node mines separately each one of these trees as soon as they are built, with minimizing the candidacy generation and without recursively building conditional sub-trees. The trees are discarded as soon as mined. Finally, all globally frequent patterns generated at each parallel node are gathered into the master node to produce the full set of frequent patterns.

The small trees we build (Co-Occurrence Frequent Item-tree, or COFI-tree for short) are similar to the conditional FP-Tree [3] 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, a counter that computes the occurrences of this item in the tree 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. Another difference, is that a COFI-tree for a given frequent item x contains only nodes labeled with items that are more frequent or as frequent as x.

In our example, if we need to mine the Inverted Matrix in Figure 2 using a two-processors machine with $\sigma > 4$, the starting line for each parallel node would be location 13, as it is the location of the first frequent item. Processor 1 finds all frequent patterns related to items at locations 13, 15, and 17 which are items (F, D, and B). Processor 2 would generate all frequent patterns related to items at locations 14, and 16 which are for items E and C.

The first Co-Occurrence Frequent Item-tree is built for item F. In this tree for F, all frequent items which are more frequent than F and share transactions with F participate in building the tree. The tree starts with the root node containing the item in question, F. For each sub-transaction containing item F with other frequent items that are more frequent than F, a branch is formed starting from the root node F. 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 COFI-trees for frequent items F and E. In Figure 3, the rectangle nodes

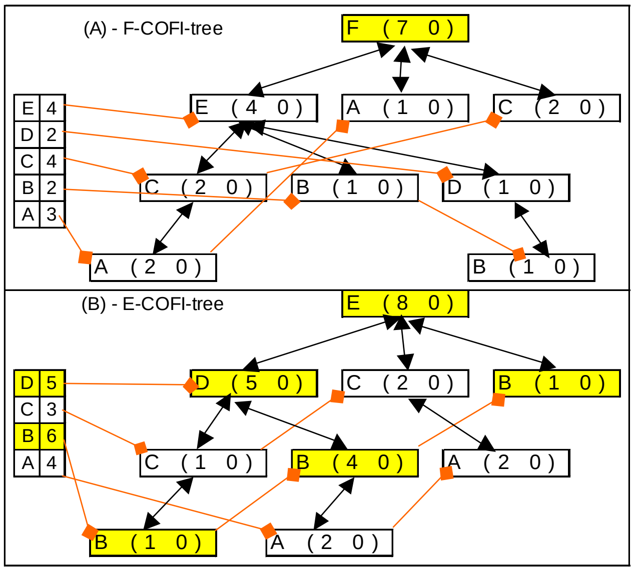


Figure 3. F and E COFI-trees

are nodes from the tree with an item label and two counters. The first counter is a support for that node while the second counter, called *participation-count*, is initialized to 0 and is used by the mining algorithm discussed later. The nodes have also pointers: 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 in the figures 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 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 assigned to processor 1 that reads from the Inverted Matrix in Figure 2 all sub-transactions that starts with item F. The first sub-transaction has items FECA, and consequently 4 nodes are created, as FECA: 1 forms one branch with support = 1 for each node in the branch. The second sub-transaction has FEB. Nodes for F and E 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 and F become 2. FA is read then, and a new node for A is created with support = 1, and the F support is incremented by 1 to become 3. FC is read twice, and a new node for item C is created with support =2, and F support becomes 5. FEDB 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 3, F becomes 6. Finally the last sub-transaction FECA is read, its branch already exist, and only the counters for all nodes are incremented by 1. The participation count for each node is initialized to 0. The header in the F-COFI-tree, like with FP-Trees, 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 COFI-tree for a frequent item x contains only nodes labeled with items that are more frequent or as frequent as x, and share at least one transaction with x. Based on this definition, if A has support greater than B then the B COFI-tree is most likely larger than the COFI-tree for item A as more items would participate in building the B COFI-tree. In other words, the higher the support of a frequent item, the smaller its COFI-tree is. We use this observation to improve the load balance between processors. We need to distribute the creation and mining of these COFI-trees among the available processors in a away that the load is distributed normally and no processor has to build large trees while others build small ones. Our strategy in distributing this load is simple: After ordering the frequent items by their support, starting from the least frequent, each processor successively receives one item. The process is repeated until all items are distributed. In other words, if we have m processors, and n COFI-trees need to be built, assuming $m < n$ then processor 1 builds the COFI-tree for the least frequent item, processor 2 builds the COFI-tree for the next least frequent item, and so on up to processor m. After that processor 1 takes item m+1 and so on until all n items are distributed.

3.1 Mining the COFI-trees

The mining process is done for each tree independently with the purpose of finding all frequent *k*-itemset patterns that the item on the root of the tree participates in. 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. Figure 4 shows the frequent itemsets containing E discovered assuming a support threshold greater than 4. Mining the “COFI-tree of item E” starts by identifying all non-frequent items with respect to the E item.

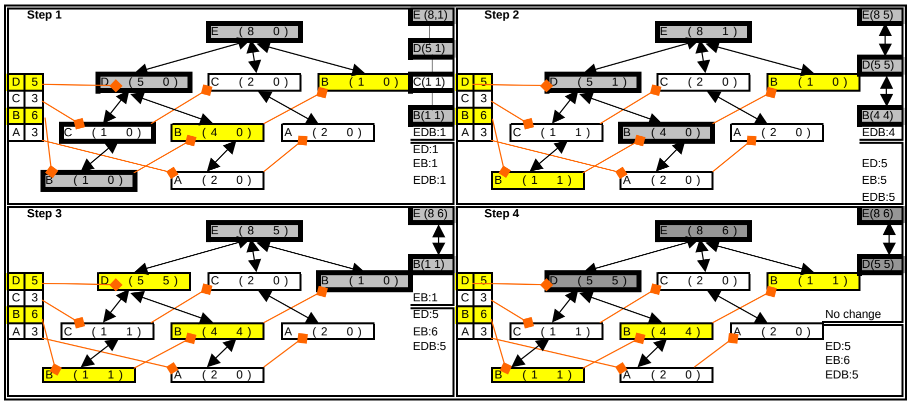


Figure 4. Steps needed to generate frequent patterns related to item E

Items A, and C occur 3 times with item E, and consequently they cannot form a frequent pattern with item E. All nodes with labels A, and C will be discarded during the mining process. After eliminating non-frequent items in this tree the mining process starts from the most frequent item in 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 equal 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 equals 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 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 is equal 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 by all processors to generate all frequent patterns.

4 Discussion and Future Work

In our research we have implemented these algorithms sequentially, and promising results were achieved[1] as we have outperformed, by more than one order of magnitude, some of the well known algorithms such as *Apriori* and FP-growth. As we have recently acquired a cluster of 8 nodes made of 16 processors, we are currently working on the parallel implementations of these algorithms as described in this paper. Our research reinforces the following: (1) In mining extremely large transactions; we should not work on algorithms that build huge memory data structures, nor on algorithms that scan the massive transactions many times. What we need is a disk-based-algorithm that can store the massive size and allow random access, and small memory structures that can be independently created and mined based on the available absolute memory size. (2) building embarrassingly parallel algorithms, which are algorithms that almost eliminate any need for communications, is the way to go. Especially for mining extremely large data sets, only communicating the frequent patterns can be considered a costly task that consumes most of the mining time. Our proposed algoritms are embarrassingly parallel.

References

[1] M. El-Hajj and O. R. Zaïane. Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining, university of alberta, technical report 08-03, March 2003.

[2] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufman, San Francisco, CA, 2001.

[3] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD*, Dallas, 2000.

[4] O. R. Zaiane, M. El-Hajj, and P. Lu. Fast parallel association rule mining without candidacy generation. In *in Proc. of the IEEE 2001 International Conference on Data Mining (ICDM01)*, San Jos, CA, USA, December 2001.