ISSN: 2229 – 6956(ONLINE)
DOI: 10.21917/ijsc.2010.0010

PARALLEL MINING OF FREQUENT MAXIMAL ITEMSETS USING ORDER PRESERVING GENERATORS

R.V. Nataraj[1] and S. Selvan[2]
[1]*Department of Information Technology, PSG College of Technology, Tamil Nadu, India*
E-mail: rv.nataraj@gmail.com
[2]*Department of Computer Science and Engineering, Alpha Engineering College, Tamil Nadu, India*
E-mail: drselvan@ieee.org

Abstract
In this paper, we propose a parallel algorithm for mining maximal itemsets. We propose POP-MAX (Parallel Order Preserving MAXimal itemset algorithm), a fast and memory efficient parallel algorithm which enumerates all the maximal patterns concurrently and independently across several nodes. Also, POP-MAX uses an efficient maximality checking technique which determines the maximality of an itemset using less number of items. To enhance the load sharing among different nodes, we have used round robin strategy which achieves load balancing as high as 90%. We have also incorporated bit-vectors and numerous optimizations to reduce the memory consumption and overall running time of the algorithm. Our comprehensive experimental analyses involving both real and synthetic datasets show that our algorithm takes less memory and less running time than other maximal itemset mining algorithms.

Keywords:
Data Mining, Closed Itemsets, Maximal Itemsets, Mining Methods

1. INTRODUCTION

The need for frequent itemset mining from transactional dataset and graph datasets has been well discussed in the literature [3][4][5][6][7] and it is a fundamental step to several data mining tasks including association rules, social network analysis, protein interaction analysis, correlation analysis and association based classification analysis. The problem is stated as follows: Given transactional dataset, enumerate all itemsets that occur in at least a user specified percentage (support) of transactions. Frequent itemsets mining is a computationally demanding task and has been an active area of research in the field of data mining. Several algorithms have been proposed in the literature including Apriori [15], FP-growth[13], Transaction Mapping Algorithm [4], ECLAT, H-Mine, Patricia Mine and FP-Growth* [5]. However, there is a major problem in mining frequent itemsets with respect to the number of result patterns that is generated i.e. the users are swarmed with too many frequent itemsets and it complicates the result analysis process. For example, if a dataset contains a frequent itemset of length x, then 2^x frequent itemsets would be generated. For large values of x, generating all frequent itemsets is computationally not possible. To overcome this, two alternatives have been proposed in the literature including closed itemset mining and maximal itemset mining. A frequent itemset is said to be closed if none of its immediate superset has the same support. Closed itemset mining exploits the Galois[19] operator and outputs small number of patterns that still contain enough information of all frequent itemsets i.e. the set of closed itemsets form a condensed representation of the set of all frequent itemsets without any loss of information. Several algorithms have been proposed to address closed itemset mining problem including A-close, FP-close [5], LCM, ECLAT-Close, AFOPT-Close [28], PG-Miner [27], Closet [20], Closet+ [21], DCI-Close [3] B-Miner and C-Miner [2]. However, for dense datasets with high pattern density, the number of closed itemsets is still large in number. The concept of maximal itemsets mining further condenses the set of closed patterns and a frequent itemset is said to be maximal if none of its immediate supersets are frequent. Several algorithms have been proposed in the literature including FP-MAX [5], AFOPT-MAX [28], GenMAX, MAFIA [6], and MAXMiner. It has been found that maximal patterns are adequate for most of the real time applications including association rules and biological applications [29]. However, the maximal itemset mining is computationally more expensive than closed itemset mining and frequent itemset mining. This is due to the fact that all maximal itemsets are unique to each other and there is no subset or superset relation between the maximal itemsets. It should be noted that all frequent itemsets and closed itemsets can be related either by subset or superset relationship. Moreover, it has been proved that the complexity class of maximal pattern mining is NP-HARD [30]. Most of the proposed algorithms for mining maximal itemsets are based on the enumeration of frequent itemsets and it outputs maximal patterns among them. Unlike closed itemset mining and frequent itemset mining, this requires the patterns to be stored in memory and hence the algorithms require more memory and computation. Some algorithms are based on maximality checking which is a computationally expensive task but require no itemsets to be stored in the main memory.

Contributions: In this paper, we propose POP-MAX algorithm which enumerates the maximal itemsets concurrently on different processors without any synchronization. Compared with previous maximal itemset mining algorithms, we have made four key contributions. First, our maximality checking strategy proposed in this paper is computationally efficient since we use only a subset of the entire set of items. Second, our algorithm requires no synchronization between processors and hence the subtasks can be executed independently without communication overhead. Third, an efficient parallelization strategy based on round robin partitioning of base itemset is proposed which achieves load balancing as high as 90% in most cases. Finally, our framework greatly enhances the mining efficiency, since all the subtasks operate in the reduced transaction space.

The rest of the paper is organized as follows. Section 2 presents the preliminaries associated with this paper. In section 3 we present our subtask generation method, the algorithm and its description while section 4 analyzes the experimental results comprehensively. In section 5, we conclude the paper.

DOI: 10.21917/ijsc.2010.0010

2. PRELIMINARIES

Let T denote a set of rows (transactions) and I denote a set of columns (items). In Boolean context, the transactional data is represented by a Boolean matrix, M, of relation Θ Í *T x I*.

Galois Connection: Let *T* Í *T, I* Í *I, f(T, M) = { i* Î *I* | " *t* Î *T, (t,i)* Î *M }* and *g(I,M)={t* Î *T* | " *i* Î *I, (t,i)* Î *M }*. The function *f* provides the set of columns (items) that are common to a set of rows (transactions) and the function *g* provides the set of rows that share a set of columns. *(f, g)* is called Galois connection between *T* and *I* and the Galois closure operators are denoted as $h = f \circ g$ and $h' = g \circ f$ [19].

Closed and Maximal Itemsets: An itemset, *I*, is said to be frequent closed itemset if *h(I)=I* and the *support(I)* ≥ *min-supp*, where *min-supp* is the user defined threshold value. An itemset, *I*, is said to be maximal frequent itemset, if *support(I)* ≥ *min-supp* and Ø$ *I'* such that *support(I')* ≥ *min-supp* and *I'* É *I*.

Order preserving closed itemset generation algorithm [3] is based on the following principle: "every closed itemset is a superset to another closed itemset". The algorithm visits the search space (item space) in the depth first manner and outputs the closed itemsets. The procedure attempts to build valid generators, which are subsets of another closed itemset and all the valid generators lead to a closed itemset. The order preserving algorithm takes three parameters as input: *closed_set*, which is initially empty, *pre_set*, which is initially empty and *post_set*, which contains all the items. *post_set* contains the set of items to be processed whereas *pre_set* contains the processed items from *post_set* that lead to valid generators. *pre_set* is updated when the recursive call returns and it does not change when the recursive call deepens. The algorithm builds all the possible generators by adding items from the *post_set* to *closed_set*. If the supporting transactions of the generator is subset to any one of the supporting transactions of the element *i* Î *pre_set*, then the generator is invalid i.e. the closed itemset of the current generator has already been generated while processing item *i*. The algorithm finds all the valid generators and then computes the closed itemsets.

Table.1. An Example Dataset, Δ, in Boolean Context

	i1	*i2*	*i3*	*i4*	*i5*	*i6*
t1	0	1	0	0	0	1
t2	0	1	1	0	0	1
t3	0	0	1	1	0	0
t4	0	1	1	1	0	1
t5	1	1	1	0	0	1
t6	1	1	1	1	1	1
t7	0	0	0	0	0	1
t8	0	1	0	1	1	0

3. PARALLEL MAXIMAL ITEMSET MINING

In this section, we first present the subtask generation framework for parallel maximal itemset mining. We then present the POP-MAX pseudo code and its description. In the first phase of POP-MAX, the item space is partitioned into non overlapping itemsets and each non-overlapping itemset is called a Base Itemset. The number of item in each base itemset is determined by the Base itemset Length (*BL*) parameter. *BL* is a function of number of available processors and the total number of items. For example, if there are *np* processors then *BL* = | □□| / *np*. The POP-MAX algorithm uses round robin partitioning strategy for creating base itemsets to achieve better load balancing among different processors. In round robin strategy, the entire itemset is ordered with respect to their individual support and the base itemsets are created by picking every k^{th} element, where k is the Base itemset Length. Once the first base itemset is filled with required number of items, the subsequent items are assigned to the second base itemset and so on. For the example dataset given in Table 1, assuming absolute support value as 2 and the number of available processors as 3, the support ordered itemset is *{ i2, i6, i3, i4, i1, i5 }* and the base itemsets are B_1= *{ i2,i4 }*, B_2=*{ i6, i1 }* and B_3=*{ i3, i5 }*. The base itemsets without using round robin partitioning strategy are B_1=*{ i2,i6 }*, B_2=*{ i3, i4 }* and B_3=*{ i1, i5 }*. After creating the required number of base itemsets, the reduced dataset is created for each of the base itemset by removing transactions which do not contain at least any one item of the corresponding base itemset. Also, the items that do not interact with any one of its other base itemset item minimum of *min-support* times is removed. This is because each subtask generates only closed itemsets which contains at least one item from its corresponding base itemset. Hence, a transaction which does not contain any one of the item from the given subtask's base itemset can be removed from the dataset of that particular subtask since that particular transaction will not support any of the closed itemsets of that subtask. This transaction reduction along with item reduction improves the mining efficiency. The reduced dataset for B_1, B_2 and B_3 is given in Table 2, Table 3 and Table 4 respectively. In Table 2, *t7* is removed since it is not supported by either *i2* or *i4*. Similarly, in Table 4 *t3* and *t8* are removed since those transactions do not contain both *i1* and *i6*. Also *i5* is removed in Table 3 since *i5* interact only once with *i6* and *i1* whereas the minimum support assumed is 2. It should be noted that this technique greatly reduces the running time of the algorithm for sparse datasets.

Table.2. Reduced dataset for B_1 Base Itemset

	i1	*i2*	*i3*	*i4*	*i5*	*i6*
t1	0	1	0	0	0	1
t2	0	1	1	0	0	1
t3	0	0	1	1	0	0
t4	0	1	1	1	0	1
t5	1	1	1	0	0	1
t6	1	1	1	1	1	1
t8	0	1	0	1	1	0

Table.3. Reduced dataset for B_2 Base Itemset

	i1	*i2*	*i3*	*i4*	*i6*
t1	0	1	0	0	1
t2	0	1	1	0	1
t4	0	1	1	1	1
t5	1	1	1	0	1
t6	1	1	1	1	1
t7	0	0	0	0	1

Table.4. Reduced dataset for B_3 Base Itemset

	i1	*i2*	*i3*	*i4*	*i5*	*i6*
t2	0	1	1	0	0	1
t3	0	0	1	1	0	0
t4	0	1	1	1	0	1
t5	1	1	1	0	0	1
t6	1	1	1	1	1	1
t8	0	1	0	1	1	0

At each subtask, we also further reduce the dataset while mining maximal itemsets and is explained as follows. If a subtask's base itemset contain *n* items, then maximal itemsets that starts with each of the *n* items are enumerated in that subtask. For example, for the base itemset B_1, the maximal itemset that starts with *i2* and *i4* are generated. While enumerating maximal itemsets that starts with a particular item, the dataset is further reduced by removing transactions that do not support that particular item. Table 5 and Table 6 show the reduced dataset of *i2* and *i4* of first subtask whereas Table 7 shows the reduced dataset for *i1* of B_2. This technique further improves the mining efficiency at each of the subtasks.

Table.5. Reduced dataset for *i2* of B_1

	i1	*i2*	*i3*	*i4*	*i5*	*i6*
t1	0	1	0	0	0	1
t2	0	1	1	0	0	1
t4	0	1	1	1	0	1
t5	1	1	1	0	0	1
t6	1	1	1	1	1	1
t8	0	1	0	1	1	0

Table.6. Reduced dataset for *i3* of B_1

	i1	*i2*	*i3*	*i4*	*i5*	*i6*
t3	0	0	1	1	0	0
t4	0	1	1	1	0	1
t6	1	1	1	1	1	1
t8	0	1	0	1	1	0

Table.7. Reduced dataset for *i1* of B_2

	i1	*i2*	*i3*	*i4*	*i6*
t5	1	1	1	0	1
t6	1	1	1	1	1

The following explains why round robin partitioning strategy results in high load balancing among different processors. It should be noted that, for a particular item *p*, the number of patterns that starts with *p* is determined by the number of elements that are present in the *post_set(p).* If *post_set(p)* contain large number of elements, then it is very likely that more patterns that start with *p* would be generated. Table 8 and Table 9 shows the *pre_set* elements and *post_set* elements for each of the subtasks without round robin strategy after mapping support ordered items to continuous integers i.e. *i2, i6, i3, i4, i1* and *i5* are mapped to *1, 2, 3, 4, 5* and *6* respectively. Table 9 shows the *pre_set* and *post_set* combination with round robin strategy. From Table 8 and Table 9, we can notice that round robin strategy equally distributes the *post_set* elements and hence results in high load balancing by distributing patterns across all the available processors.

Table.8. *pre_set* and *post_set* combination without round robin strategy

	{pre_set} {post_set}
Subtask 1	{ } { 1 2 3 4 5 6 } { 1 } { 2 3 4 5 6 }
Subtask 2	{ 1 2 } { 3 4 5 6 } { 1 2 3 } { 4 5 6 }
Subtask 3	{ 1 2 3 4 } { 5 6 } { 1 2 3 4 5 } { 6 }

Table.9. *pre_set* and *post_set* combination with round robin strategy

	{pre_set} {post_set}
Subtask 1	{ } { 1 2 3 4 5 6 } { 1 2 3 } { 4 5 6 }
Subtask 2	{ 1 } { 2 3 4 5 6 } { 1 2 3 4 } { 5 6 }
Subtask 3	{ 1 2 } { 3 4 5 6 } { 1 2 3 4 5 } { 6 }

3.1 POP-MAX PSEUDO CODE

INPUT: Dataset, Δ, support value and np, number of processors
OUTPUT: Set of maximal patterns satisfying the support constraint.

1. *Compute Φ1 (frequent 1 items) from Δ*
2. *Sort the items of Φ1 in its support descending order and map the items to continuous integer space*
3. *//Generate the Base itemsets using round robin partitioning*
4. *BL=| Φ1 | / np*
5. *for (i=1; i<=np; i++)*
6. *for(j=0; j<BL; j++)*
7. *$B_i = i + (np * j) \cup B_i$*
8. *endfor*
9. *endfor*
10. *//Generate the reduced transaction set for Base Itemsets*
11. *for (k=1; k<=np; k++)*
12. *T_k={ t ∈ T | ∃ i∈ B_k, (t,i)=1 }*
13. *call Mine_Maximal_itemsets(Bi,Ti) on k^{th} Processor*
14. *endfor*
15. *Mine_Maximal_Itemsets$_i$ (B_i, T_i)*
16. *{*
17. *" i Î B_i (i^{th} subtask base itemset)*
18. *pre_set = { i' Î F1 | i' f i }*
19. *closed_set = null*

DOI: 10.21917/ijsc.2010.0010

```
20.      post_set =i ∪{ i' Î F1 | i' p  i }
21.      reorder the vertical bit-vector space such that
     supporting transactions of i are consecutive
          in its bit-vector space.
22.      tid_set_c = { t ∈ T_i | (t,i)=1 }
23.      OP-MAX*_i(post_set, closed_set, pre_set, tid_set_c )
24.  }

25.  OP-MAX*_i( post_set, closed_set, pre_set, tid_set_c )
26.  {
27.     while (post_set!=null)
28.  z:   i''=min(post_set)
29.      tid_set_g = tid_set_c ∩ g(i'')
30.      if | tid_set_g |>min_support &&
           ( " jÎ pre_set,  tid_set_g Ë g(j))
31.         write closed_set, post_set,
                 pre_set ∪ i'', tid_set_c to stack
32.         closed_set=closed_set ∪ i''
33.         " kÎ post_set
34.            if tid_set_g Í  g(k)
35.               closed_set=closed_set ∪ k
36.               post_set=post_set \ k
37.            endif
38.         tid_set_c = tid_set_g
39.         write closed_set to disk
40.      else
41.         if   (post_set!=null)  goto z:   endif
42.      endif
43.      if  (post_set==null && stack is not empty)
44.            if  $ rÎ pre_set  and
                  support(closed_set ∪ r) > minsupp
45.                  discard the closed_set
46.            else
47.                  output the closed_set as maximal itemset
48.            endif
49.            pop from stack to closed_set, preset,
                              post_set and tid_set_c
50.      endif
51.      if (closed_set==null)
52.            return
53.      endif
54.    endwhile
55.  }
```

4. IMPLEMENTATION AND RESULT ANALYSIS

We have implemented our algorithm using C language and the code was compiled using 32 bit Microsoft Visual C++ compiler. We have written our own stub code to execute the subtasks on different processors. Our implementation of POP-MAX is as follows. We have used bit-vectors to represent the dataset in main memory. Before the subtasks are created, we compute the frequent-1-items and the items are sorted with respect to their support and mapped to continuous integers for ease of processing. All the processing is done in the mapped space and we remap the items while writing the maximal itemsets to disk. We have used user defined stack to store the information required for backtracking and each stack element contains *closed_set* and the associated *post_set*, *pre_set* and *tid_set*. The *tid_set* is a bit-vector and we use bit-wise AND operations for closure computation and duplicate checking. While creating *post_set* and *pre_set*, we adopt a particular ordering strategy proposed in [7] to speed up the computation by reducing the bit-wise operations required for closure checking and duplicate detection i.e. the *pre_set* contains item with higher support and the *post_set* contains items with lower support and all the *pre_set* items are ordered in descending order of their support whereas the items in the *post_set* are ordered in ascending order with respect to their support. This ordering strategy facilitates fast duplicate checking and also improves the efficiency of closure checking.

We have used four sets of data in our experiments and the characteristics of the datasets are given in Table 10. The synthetic dataset generator is downloaded from Illimine project's website. In the notation TxIyDzNq, x indicates the average transaction length, y indicates the average pattern length, z indicates the total number of row (transaction) instances and N indicates the total columns (items). In all the datasets, columns are separated by space and rows are separated by a space and new line character. All the experiments were conducted on an isolated pentium 4 machine with 1GB main memory loaded with windows XP operating system.

Table.10. Datasets used

Dataset	# Items	#Transactions
Chess	72	3126
Gazelle	467	59601
Pumsb_star	7117	49046
Connect	129	67557
T25I20D10K	1000	10000
T10K4D10K	1000	10000

To get the accurate time to the extent possible, while conducting experiments, we have made sure that no other programs were running in the background. All times shown include time for reading data from disk and generating all the patterns satisfying the given constraints. To find the accurate peak main memory usage and peak page file usage, we have not used any specialized software since it incurs much overhead. We have written a small windows kernel based C program using windows process library API that will fetch the main memory usage statistics whenever a process is terminated. Since we extract the needed information from the windows kernel itself, the load made by this program on the memory and the processor is completely negligible. We have used the concept of mean and standard deviation, for calculating the load sharing percentage achieved among different subtasks. We have computed the actual standard deviation and maximal standard deviation for the time taken by each of the subtasks. It should be noted that if all

the subtasks take equal amount of time, then the actual standard deviation value is zero and the maximum standard deviation occur if one process takes all the running time. Also, the actual standard deviation value is always less than maximal standard deviation value. Hence, we compute the load sharing percentage as follows: $(1-(A_{SD}/M_{SD})*100)$ where A_{SD} is the actual standard deviation and M_{SD} is the maximal standard deviation. It is to be noted that, the lower the value of A_{SD} is, higher will be the load sharing among different processors. We have done a large number of experiments and shall present only representative results here. The results shown in Table 11 compare the load sharing percentage of POP-MAX algorithm with and without round robin strategy for chess dataset. As shown, the round robin partitioning strategy on the average achieves 4 times better load sharing when executed with 4 processors. Similarly, Table 12 shows the result of connect dataset which shows that the round robin strategy achieves 2.5 times better load balancing on the average. Table 13 presents the results obtained for gazelle dataset whereas Table 14 presents the number of maximal patterns that are generated by each of the subtasks. As shown in the results, the round robin strategy efficiently distributes the patterns across different subtasks and hence reduces the overall running time. Table 15 shows the result of pumb_star dataset whereas Table 16-18 shows the result obtained from two synthetic datasets. Table 19 compares the memory usage of POP-MAX with other algorithms and to make the comparison fair, we generated only one subtask because the other algorithms are not parallel algorithms. The results clearly indicate that POP-MAX takes less memory that the other algorithms for its execution.

Table.11. Load Sharing Percentage among different subtasks for Chess dataset

support	Load Sharing Percentage of POP-MAX			
	Without round robin Strategy		With round robin strategy	
	4 Processors	8 Processors	4 Processors	8 Processors
127	21.57	51.14	92.47	72.18
95	19.14	41.46	90.95	83.37
63	13.4	41.35	85.38	80.46
32	18.85	43.22	87.14	78.27

Table.12. Load Sharing Percentage for connect dataset

support	Load Sharing Percentage of POP-MAX			
	Without round robin Strategy		With round robin strategy	
	4 Processors	8 Processors	4 Processors	8 Processors
33778	33.87	38.79	83.44	52.71
30400	21.74	27.03	75.82	43.25
27022	35.9	41.47	69.82	65.45
23644	24.17	38.56	64.33	64.41
20267	42.94	41.15	70.58	66.77
16889	24.19	45.68	83.44	65.82

Table.13. Load Sharing Percentage among different subtasks for gazelle dataset

support	Load Sharing Percentage of POP-MAX			
	Without round robin Strategy		With round robin strategy	
	4 Processors	8 Processors	4 Processors	8 Processors
36	50.89	65.94	83.37	89.58
30	56.91	69.89	83.55	91.20
24	52.92	68.66	87.94	90.83
18	42.92	62.29	82.83	89.26
12	26.96	52.21	84.83	87.25
6	38.95	52.84	93.53	86.96
1	41.65	55.48	86.9	89.06

Table.14. Pattern distribution among different subtasks for gazelle dataset

support	Distribution of Maximal Itemsets among Different Subtasks			
	Subtask 1	Subtask 2	Subtask 3	Subtask 4
	Without Round Robin Strategy			
24	7537	3455	1618	346
18	15065	5925	1071	166
12	57589	24116	2880	248
6	57607	62591	8632	924
1	85	924	1253	416
	With Round Robin Strategy			
24	3413	3060	4519	1964
18	5375	5996	7060	3796
12	18307	23831	20642	22053
6	28685	28526	28920	43623
1	651	582	650	795

Table.15. Load Sharing Percentage for pumsb_star dataset

Supp-ort	Load Sharing Percentage of POP-MAX			
	Without round robin Strategy		With round robin strategy	
	4 Processors	8 Processors	4 Processors	8 Processors
19618	81.6	73.50	84.84	78.76
17166	83.61	80.41	85.41	85.13
14713	74.48	74.58	85.03	77.08
12261	73.87	71.35	80.56	79.13
9809	56.83	65.46	92.92	86.98
7356	48.53	63.80	92.48	79.57
4904	48.06	53.56	82.95	81.24

DOI: 10.21917/ijsc.2010.0010

Table.16. Load Sharing Percentage for T10I4D10K dataset

support	Load Sharing Percentage of POP-MAX	
	Without round robin Strategy	With round robin strategy
	4 processors	4 Processors
7	48.22	83.93
6	45.85	84.54
5	42.02	84.79
4	38.87	86.61
3	37.75	88.97
2	36.79	89.06
1	58.72	95.01

Table.17. Load Sharing Percentage for T25I20D10K dataset

support	Load Sharing Percentage of POP-MAX	
	Without round robin Strategy	With round robin strategy
	4 processors	4 Processors
7	25.38	87.1
6	27.95	88.38
5	30.96	90.22
4	34.04	91.92
3	37.98	93.15
2	48.01	95.53
1	56.16	96.98

Table.18. Pattern distribution among different subtasks for T25I20D10K dataset

support	Distribution of Maximal Itemsets among Different Subtasks			
	Subtask 1	Subtask 2	Subtask 3	Subtask 4
	Without Round Robin Strategy			
24	411454	295032	88685	14623
18	561004	452558	129676	19493
12	842403	795657	242704	32560
6	598193	884206	358835	55820
1	0	47	1486	3855
	With Round Robin Strategy			
24	176531	212492	223088	197685
18	260220	299406	313550	289554
12	436732	492220	507202	477168
6	453326	463381	493851	486493
1	1127	1480	1540	1241

Table.19. Peak main memory usage in bytes for gazelle dataset

Supp-ort	Peak Main Memory Usage			
	FP-MAX	AFOPT	POP-MAX	LCM-MAX
24	4341760	3497984	2629632	5287936
18	4079616	3444736	2711552	5292032
12	4288512	4046848	2809856	5300224
6	5623808	9523200	2936832	5304320
1	12066816	15245312	3166208	5341184

5. CONCLUSION

Efficient mining of maximal itemset mining is a fundamental task to several data mining applications and we have proposed a fast and memory efficient parallel algorithm in this paper. It adaptively create subtasks using round robin strategy which achieves very high load sharing among different processors. Efficient maximality checking strategy was presented which greatly improves the algorithm performance. We are further investigating techniques to further reduce the overall running time of the algorithm.

ACKNOWLEDGEMENTS

We wish to thank C. Lucchese, Liping Ji and Guimei Liu for responding to our queries. We thank J. Han, advisor of Illimine project, for providing the synthetic dataset generator and supporting the website. We also thank Prof. Bart Goethals for supporting FIMI website.

REFERENCES

[1] Jinyan Li, Guimei Liu, Haiquan Li, Limsoon Wong. 2007. "Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms". *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 12, pp. 1625-1637.

[2] Liping Ji, Kian-Lee Tan,K H. Tung. 2007. "Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets". *IEEE Trans. on Knowledge and Data Engineering*. Vol.19, No.9.

[3] C. Lucchese,S. Orlando and R. Perego. 2006. "Fast and Memory Efficient Mining of Frequent Closed Itemsets". *IEEE Transactions on Knowledge and Data Engineering*. Vol.18, No 1, pp. 21-36.

[4] Mingjun Song, Sanguthevar Rajasekaran. 2006. "A Transaction Mapping Algorithm for Frequent Itemsets Mining". *IEEE Transactions on Knowledge and Data Engineering.* Vol.18, No.4, pp.472-481.

[5] G. Grahne and J.Zhu. 2005. "Fast Algorithms for Frequent Itemset Mining Using FP-Trees". *IEEE Transactions on Knowledge and Data Engineering*. Vol.17, No.10, pp.1347-1362.

[6] D. Burdick, M.Calimlim, J.Flannick, J.Gehrke, Y.Yiu. 2005. "MAFIA: A Maximal Frequent Itemset Algorithm". *IEEE Transactions on Knowledge and Data Engineering*, Vol.17, No.11, pp.1490 – 1504.

[7] S. Selvan and R V Nataraj. 2008. "Efficient Mining of Maximal Patterns using Order Preserving Generators", *in proc. 16th Intl. Conf. on Advanced Computing and Communications*, Chennai, India.

[8] J. Besson, C. Robardet, J.F. Boulicaut and S. Rome. 2005. "Constraint Based Concept Mining and its Application to Microarray Data Analysis". *Journal of Intelligent Data Analysis*, pp. 59-82.

[9] Ji Liping. 2006. "Mining Localized Co-expressed Gene Patterns from Microarray Data". PhD dissertation, School of Computing., National University of Singapore.

[10] Ji Liping, K.L. Tan and A.K.H. Tung. 2006. "Mining Frequent Closed Cubes in 3D datasets". *Proc. 32nd int. conference on very large databases.*

[11] Gao Cong, Kian-Lee Tan, A. K. H. Tung, Feng Pan. 2004. "Mining Frequent Closed Patterns in Microarray Data". *ICDM'04*, Vol.1, No.4, pp.363-366.

[12] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, 2004 "Mining Frequent Pattern without candidate Generation:A Frequent Pattern Approach" Journal of Data Mining and Knowledge Discovery , Springer, Vol.8, No.1, pp.53-87.

[13] J. Han, J. Pei, and Y. Yin. 2000. "Mining frequent patterns without candidate generation". *In Proceeding of Special Interest Group on Management of Data*, pp.1-12.

[14] R. Agrawal, T. Imielinski, and A. Swami. 1993. "Mining association rules between sets of items in large Databases". *In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp.207-216.

[15] R. Agrawal and R. Srikant. 1994. "Fast algorithms for mining association rules". *In Proceeding of Int. Conf. Very Large Data Bases*, pp.487-499.

[16] A. Savasere, E. Omiecinski, and S. Navath. 1995. "An efficient algorithm for mining association rules in large databases". *In Proc. of Intl. Conf. on Very Large Databases (VLDB).*

[17] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. 1997. "Dynamic itemset counting and implication rules for market basket data". *SIGMOD Record,* Vol.6, No.2, pp.255-264.

[18] R. Agrawal and J. C. Shafer. 1996. "Parallel mining of association rules". *IEEE Transactions on Knowledge and Data Engineering*, Vol.8, No.6, pp.962–969.

[19] N. Pasquier, Y. Bastide, R. Taouil, and L.Lakhal. 1999. "Discovering Frequent Closed Itemsets for Association Rules". Proc. 7th Int. Conf. Database Theory (ICDT'99), pp.398-416.

[20] J.pei, J.Han, and R. Mao. 2000. CLOSET: "An Efficient Algorithm for mining Frequent Closed Itemsets", *proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)* pp.11-20.

[21] J. Wang, J.Han and J.pei. 2003. "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets". *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp.236-245.

[22] M.J. Zaki, C.J.Hsiao. "CHARM: An Efficient Algorithm for Closed Itemset Mining. *Proc SIAM Int. Conf. Data Mining"*, pp.457-473.

[23] M.J. Zaki, C.J. Hsiao. 2005, "Efficient Algorithms for Mining Closed Itemsets and their Lattice Structure". *IEEE Trans. on Knowledge and data Engineering*, Vol.17, No.4, pp.462-478.

[24] Ahmed Shakil, Frans Coenen, Paul Leng. 2006. "Tree based partitioning of data for association Rule Mining". *Knowledge and Information Systems, Springer,* Vol.10, no.3, pp.315-331.

[25] A. Veloso, M. Otey, S. Parthasarathy, and W. Meira. 2003. "Parallel and distributed frequent itemset mining on dynamic datasets". *In Proc. of the High Performance Computing Conference*, HiPC, Hyderabad, India.

[26] Yew-kwong Woon, Wee-keong Ng, Ee-Peng Lim. 2004. "A Support-ordered Trie for Fast Frequent Itemset Discovery". *IEEE Transactions on Knowledge and Data Engineering*, Vol.16, No.7, pp.875-879.

[27] H.D.K. Moonesinghe, Samah Fodeh, P. N. Tan. 2006. "Frequent Closed Itemset Mining Using Prefix Graphs with an Efficient Flow-Based Pruning Strategy", *ICDM'06*.

[28] Guimei Liu. 2005. "Supporting Efficient and Scalable Frequent Pattern Mining", PhD Thesis, Hong Kong University.

[29] I. Rigoutsos and A. Floratos. 1998. "Combinatorial Pattern Discovery in Biological Sequences: The Teiresias Algorithm", Bioinformatics, Vol.14, pp.55-67.

[30] Guizhen Yang. 2004. "The complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns". KDD'04, Seattle, Washington.