computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

PARALLEL MINING OF LARGE MAXIMAL BICLIQUES USING ORDER PRESERVING GENERATORS

R.V. Nataraj [1], S. Selvan [2]

[1] PSG College of Technology, India, rvn@ieee.org, www.psgtech.edu
[2] Francis Xavier Engg. College, India, drselvan@ieee.org

Abstract: *In this paper, we propose a parallel algorithm for mining large maximal bicliques from graph datasets. We propose POP-MBC (Parallel Order Preserving Maximal BiClique mining algorithm), a fast and memory efficient parallel algorithm, which enumerates all the maximal bicliques independently and concurrently across several processors without any synchronization between the processors. The POP-MBC algorithm is highly memory efficient since it does not store the previously computed patterns in the main memory and requires only the dataset to be stored in the memory. To enhance the load sharing among different nodes, POP-MBC uses a round robin strategy which enables to achieve load balancing as high as 90%. We have also incorporated bit-vectors and numerous optimization techniques exploiting the symmetric property of the graph dataset to reduce the memory consumption and overall running time of the algorithm. Our comprehensive experimental analyses involving publicly available datasets show that our algorithm distributes the load among the different processors equally and takes less memory, less running time than other maximal biclique mining algorithms.*

Keywords: *Data mining, Knowledge Discovery, Maximal Bicliques, Mining Methods.*

1. INTRODUCTION

The need for maximal biclique (complete bipartite subgraph) mining from graph datasets has been well discussed in the literature [1][4][5][6][8] and has several applications in the field of data mining including social network analysis and protein interaction network analysis [1]. The problem is stated as follows: Given a graph containing vertices interaction, enumerate all the maximal row vertices and column vertices pair of user specified size such that each of the row vertices interacts with all the column vertices. Large maximal biclique mining is a computationally demanding task and requires more computational resources to enumerate all the bicliques of the given dataset and has been an active area of research. Moreover, the complexity class of maximal edge biclique problem has been proven to be in NP-Complete [5]. Several algorithms have been proposed in the literature including MICA [4], Eppstein Algorithm [8] and LCM-MBC[1]. Eppstein's algorithm (induced maximal biclique) has a complexity of $O(a^3\ 2^{2a}\ n)$ where n is the number of vertices and a is the arboricity of the graph. Alex et al's MICA algorithm (non induced maximal bicliques) has a time complexity of $O(n^2$ x $N)$ and a space complexity of $O(N)$ where N is the number of maximal bicliques. The major drawback of MICA algorithm is that it requires the entire generated maximal biclique subgraphs to be stored in main memory for duplicate detection and runs out of memory for most of the executions with large graph datasets. Recently, the relationship between closed patterns and maximal bicliques is reported in the literature and closed pattern mining algorithms are extended to enumerate maximal bicliques. G. Liu et. al, extended LCM algorithm and proposed LCM-MBC algorithm [1] for maximal biclique enumeration. Unlike MICA, the LCM-MBC algorithm does not store the already computed bicliques in memory and hence the algorithm never runs out of main memory. The time complexity of LCM-MBC is $O(mn$ x $N)$ and the space complexity is $O(mn)$, where m is the number of edges of the graph. LCM-MBC algorithm is based on the following properties: (i) the number of closed patterns of an symmetric adjacency matrix is even; (ii) for every maximal biclique subgraph, there exists a unique closed pattern pair which corresponds to the vertex sets. The draw back of LCM-MBC algorithm is that it is sequential and takes large amount of running time for large and dense graph datasets. It should be noted that closed pattern mining in Boolean context is tractable if at least one of the dimensions (either row or column) is small with less pattern density [13]. Otherwise, the mining task is extremely hard, i.e. the enumeration of all patterns take huge amount of running time (for a particular real dense dataset, the enumeration of all patterns took 6-weeks [3] of running time). For graph datasets, the number of rows and columns are equal and for large dense datasets the running time of an algorithm can be reduced only through parallelized execution

Contributions: In this paper, we propose POP-MBC algorithm which enumerates the large maximal bicliques concurrently on several processors without any synchronization. Compared with previous maximal biclique mining algorithms, we have made three key contributions. First, our algorithm does not store the previously computed patterns in the main memory for duplicate detection and hence the algorithm is memory efficient. 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 vertex set is proposed which achieves load balancing as high as 90% in most cases.

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.

2. PRELIMINARIES

Let ς_P denote a set of row vertices and ς_X denote a set of column vertices. In Boolean context, the graph dataset is represented by a Boolean matrix, M, of relation Θ H ς_P x ς_X.

Galois Connection: Let R H ς_P, C H ς_X, $f(R, M) = \{ c$ O ς_X | " r O $R, (r,c)$ O $\Theta \}$ and $g(C,M)=\{r$ O ς_P | " c O $C, (r,c)$ O$\Theta \}$. The function f provides a set of columns that are common to a set of rows and the function g provides a set of rows that share a set of columns. *(f, g)* is called Galois connection between R and C and the Galois closure operators are denoted as $h = f \circ g$ and $h' = g \circ f$ [22].

Maximal Bicliques: *(R : C)*, where R H ς_P and C H ς_X, is called a maximal biclique in M when $R=g(C)$ in M and $C=f(R)$ in M. A maximal biclique is thus a maximal 1-rectangle in the Boolean matrix of the given dataset. For more details on closure operators, closed patterns and maximal bicliques, readers may refer [1].

Large Maximal Bicliques: A maximal biclique is said to be of size (p, q) if $|R| \geq p$ and $|C| \geq q$, where p and q are user specified size constraints.

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 O *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. In the context of graph’s adjacency matrix, an itemset may refer to row vertex set or column vertex set.

Table 1. An Example Graph dataset, Δ, in Boolean Context

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v1	0	1	0	0	0	1	1	1	1
v2	1	0	1	0	0	1	1	1	0
v3	0	1	0	1	0	0	0	0	0
v4	0	0	1	0	0	1	0	1	1
v5	0	0	0	0	0	1	0	0	1
v6	1	1	0	1	1	0	1	1	0
v7	1	1	0	0	0	1	0	0	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

3. PARALLEL MAXIMAL BICLIQUE MINING

In this section, we first present the subtask generation framework for parallel maximal biclique mining. We then present the POP-MBC pseudo code and its description. In the first phase of POP-MBC, the vertex space is partitioned into non overlapping vertex sets and each non-overlapping vertex set is called a Base vertex set. The number of items in each base vertex set is determined by the Base vertex set Length (*BL*) parameter. *BL* is a function of number of available processors and the total number of vertices. For example, if there are *np* processors then $BL = | \varsigma | / np$. The POP-MBC algorithm uses a round robin partitioning strategy for creating base vertex sets to achieve better load balancing among the different processors. In round robin strategy, the entire column vertex set is ordered with respect to their individual row support and the base vertex sets are created by picking every k^{th} vertex, where k is the Base vertex set Length. Once the first base vertex set is filled with required number of items, the subsequent vertices are assigned to the second base vertex set and so on. For the example dataset given in Table 1, assuming absolute row support value as 2 and the number of available processors as 3, the row support ordered vertex set is *{ v6, v1, v2, v8, v4, v9, v7, v3, v5 }* and the base vertex sets are B_1*={ v6 v8 v7 }* B_2 *= { v1 v4 v3 }* and B_3*= {v2 v9 v5}*. The base vertex sets without using round robin partitioning strategy are B_1*={ v6 v1 v2 }* B_2 *= { v8 v4 v9 }* and B_3*= {v7 v3 v5}*. After creating the required number of base vertex sets, the reduced vertex space is created for each of the base vertex sets by removing row vertices which do not contain at least any one item of the corresponding base vertex set. Also, the row vertices that do not interact with any one of its other base vertex item for a minimum of q number of times and the column vertices that do not interact with any one of its row vertex for a minimum of p number times are removed. It is to be noted that, p is the minimum row size and q is the minimum column vertex set size. This is because, each subtask generates only patterns which contain at least one item from its corresponding base vertex set. Hence, a row vertex which does not interact with any one of the item from the given subtask's base vertex set can be removed from the dataset of that particular subtask since that particular row vertex will not support any of the column patterns of that subtask. This vertex space 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, *v3* row vertex is removed since it is not supported by *v6, v7* and *v8* column vertices. Similarly, in Table 4, *v2* is removed since *v2* does not interact with v2, v5 and v9 column vertices. 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 vertex set

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v1	0	1	0	0	0	1	1	1	1
v2	1	0	1	0	0	1	1	1	0
v4	0	0	1	0	0	1	0	1	1
v5	0	0	0	0	0	1	0	0	1
v6	1	1	0	1	1	0	1	1	0
v7	1	1	0	0	0	1	0	0	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

Table 3. Reduced dataset for B$_2$ Base vertex set

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v2	1	0	1	0	0	1	1	1	0
v3	0	1	0	1	0	0	0	0	0
v4	0	0	1	0	0	1	0	1	1
v6	1	1	0	1	1	0	1	1	0
v7	1	1	0	0	0	1	0	0	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

Table 4. Reduced dataset for B$_3$ Base vertex set

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v1	0	1	0	0	0	1	1	1	1
v3	0	1	0	1	0	0	0	0	0
v4	0	0	1	0	0	1	0	1	1
v5	0	0	0	0	0	1	0	0	1
v6	1	1	0	1	1	0	1	1	0
v7	1	1	0	0	0	1	0	0	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

At each subtask, we also further reduce the dataset while mining maximal biclique patterns as explained below. If a subtask's base vertex set contain n vertices, then maximal bicliques that start with each of the n vertices in their column vertex set are enumerated in that subtask. For example, for the base vertex set B$_1$, the maximal biclique that starts with *v6, v7* and *v8* are generated. While enumerating maximal bicliques that start with a particular vertex, the dataset is further reduced by removing row vertices that do not support that particular item. Table 5 and Table 6 show the reduced dataset of *v1* and *v4* of second subtask whereas Table 7 shows the reduced dataset for *v5* of B_3. This technique further improves the mining efficiency at each of the subtasks.

Table 5. Reduced dataset for *v1* of B$_2$

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v2	1	0	1	0	0	1	1	1	0
v6	1	1	0	1	1	0	1	1	0
v7	1	1	0	0	0	1	0	0	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

Table 6. Reduced dataset for *v4* of B$_2$

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v3	0	1	0	1	0	0	0	0	0
v6	1	1	0	1	1	0	1	1	0
v8	1	1	0	1	0	1	0	0	1
v9	1	0	0	1	1	0	0	1	0

Table 7. Reduced dataset for *v5* of B_3

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*	*v9*
v6	1	1	0	1	1	0	1	1	0
v9	1	0	0	1	1	0	0	1	0

The following explains why round robin partitioning strategy results in high load balancing among different processors. It should be noted that, for a particular vertex *c*, the number of maximal bicliques that start with *c* in their column vertex set is determined by the number of elements that are present in the *post_set(c)[3]*. If *post_set(c)* contain large number of elements, then it is very likely that more patterns that start with *c* would be generated. Table 8 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. *v6, v1, v2, v8, v4, v9, v7, v3* and *v5* are mapped to *1, 2, 3, 4, 5, 6, 7* and *8* 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 maximal biclique patterns across all the available processors.

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

Subtask 1	*{pre_set} {post_set}*
	{ } { 1 2 3 4 5 6 7 8 9}
	{ 1 } { 2 3 4 5 6 7 8 9}
	{ 1 2 } { 3 4 5 6 7 8 9}

Subtask 2	*{pre_set} {post_set}*
	{ 1 2 3 } { 4 5 6 7 8 9}
	{ 1 2 3 4 } { 5 6 7 8 9}
	{ 1 2 3 4 5 } { 6 7 8 9}

Subtask 3	*{pre_set} {post_set}*
	{ 1 2 3 4 5 6 } { 7 8 9}
	{ 1 2 3 4 5 6 7 } { 8 9}
	{ 1 2 3 4 5 6 7 8 } { 9}

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

Subtask 1	*{pre_set} {post_set}*
	{ } { 1 2 3 4 5 6 7 8 9}
	{ 1 2 3 } { 4 5 6 7 8 9}
	{ 1 2 3 4 5 6 } { 7 8 9}

Subtask 2	*{pre_set} {post_set}*
	{ 1 } { 2 3 4 5 6 7 8 9}
	{ 1 2 3 4 } { 5 6 7 8 9}
	{ 1 2 3 4 5 6 7 } { 8 9}

Subtask 3	*{pre_set} {post_set}*
	{ 1 2 } { 3 4 5 6 7 8 9}
	{ 1 2 3 4 5 } { 6 7 8 9}
	{ 1 2 3 4 5 6 7 8 } { 9}

3.1. POP-MBC PSEUDO CODE

INPUT: Dataset, Δ, p,q constraints and np, number of processors

OUTPUT: Set of maximal bicliques satisfying the p,q size constraint.

```
Compute Φ1 (set of column vertices of support) from Δ
Sort the items of Φ1 in their row support descending
    order and map the items to continuous integer space
//Generate the Base vertex set using round robin
    partitioning
BL=| Φ1 | / np
for (i=1; i<=np; i++)
    for(j=0; j<BL; j++)
        $B_i$ = i +(np * j) ∪ $B_i$
    endfor
endfor
//Generate the reduced datasets for Base vertex sets
for (k=1; k<=np; k++)
    $R_k$={ r ∈ ςP | ∃ c ∈ $B_k$, (r,c)=1 }
    call Mine_Maximal_Bicliques($B_i$,$R_k$) on $k^{th}$
Processor
endfor

Mine_Maximal_Bicliques$_i$ ($B_i$, $R_i$)
{
    " c O $B_i$ ($i^{th}$ subtask base vertex set)
     pre_set = { c' OF1 | c' f  c }
     closed_set = null
     post_set =c ∪ { c' OF1 | c' p  c }
     reorder the vertical bit-vector space such that
       supporting transactions of i are consecutive
       in its bit-vector space.
     row_set$_c$ = { r ∈ $R_i$ |  (r,c)=1 }
     Execute$_i$(post_set, closed_set, pre_set, row_set$_c$)
}

Execute$_i$(post_set, closed_set, pre_set, row_set$_c$)
{
    while (post_set!=null)
z:     c''=min(post_set)
       row_set$_g$ = row_set$_c$ ∩ g(c'')
       if |row_set$_g$ |>p &&
          (" j Opre_set,  row_set$_g$ Л g(j))
          write closed_set, post_set,
                pre_set ∪ c'', row_set$_c$ to stack
          closed_set=closed_set ∪  c''
          " kOpost_set
             if row_set$_g$ H  g(k)
                closed_set=closed_set ∪ k
                post_set=post_set \k
             endif
          row_set$_c$ = row_set$_g$
          if |closed_set| > q
             write row_set$_g$:closed_set to disk
          endif
       else
           if   (post_set!=null)  goto z:   endif
       endif
```

```
        if (post_set==null && stack is not empty)
                pop from stack to closed_set, pre_set,
                                        post_set and $row\_set_c$
        endif
        if (closed_set==null)
                return
        endif
    endwhile
}
```

4. IMPLEMENTATION AND RESULT ANALYSIS

We have implemented our algorithm using C language and the code is 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-MBC 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-column vertices and the elements are sorted with respect to their row support and mapped to continuous integers for ease of processing. All the processing is done in the mapped space and we remap the vertices while writing the maximal bicliques to disk. We have used user defined stack to store the information required for backtracking and each stack element contains *closed_set* and its associated *post_set*, *pre_set* and *row_set*. The *row_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 [12] to speed up the computation by reducing the bit-wise operations required for closure checking and duplicate detection i.e. the *pre_set* contain items with higher support and the *post_set* contain items with lower support and all the *pre_set* items are ordered in ascending order of their support whereas the items in the *post_set* are ordered in descending order with respect to their support. This ordering strategy facilitates fast duplicate checking and also improves the efficiency of closure checking.

The datasets used in our experiments and their characteristics are given in Table 10. All the datasets are downloaded from DIMACS website (ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks). 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	# vertices	#edges
c-fat200-1	200	1534
c-fat200-2	200	3235
hamming6-2	64	1824
c-fat500-2	500	9139
Johnson16-2-4	120	5460

To get the accurate time to the extent possible, we have made sure that no other programs were running in the background while conducting the experiments. All times shown include time for reading data from disk and generating all the patterns satisfying the given size 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 occurs, if one processor 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, the 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-MBC algorithm with and without round robin strategy for c-fat200-1 dataset. As shown, the round robin partitioning strategy achieves better load sharing among different processors. Similarly, Table 12 shows the result of hamming6-2 dataset which shows that the round robin strategy achieves better load balancing on the average. Table 13 presents the results obtained for c-fat200-2 dataset whereas Table 14 presents the number of maximal biclique patterns that are generated by each of the subtasks. As shown in the results, the round robin strategy distributes the patterns across different subtasks and hence reduces the overall running time. Table 15 shows the result of c-fat500-2 dataset whereas Table 16 and Table 17 show the results obtained from johnson16-2-4 datasets. Table 18 compares the memory usage of POP-MBC with LCM-MBC and to make the comparison fair, we have generated only one subtask because the other algorithm is not a parallel algorithm. The results clearly indicate that POP-MBC takes less memory and less running time than

LCM-MBC algorithm. We have not compared with MICA algorithm since LCM-MBC was proven to be faster than MICA in [1].

Table 11. Load Sharing Percentage among different subtasks for c-fat200-1 dataset

	Load Sharing Percentage of POP-MBC			
	Without round robin strategy		With round robin strategy	
size	4 Processors	8 Processors	4 Processors	8 Processors
1	23.26	52.62	69.35	87.24
2	26.8	56.39	74.44	87.10
3	43.22	56.89	77.89	79.94
4	33.38	54.67	79.41	74.33
5	51.16	58.03	80.6	78.13
6	20.41	29.45	98	62.16

Table 12. Load Sharing Percentage for hamming6-2 dataset

	Load Sharing Percentage of POP-MBC			
	Without round robin strategy		With round robin strategy	
Size	4 Processors	8 Processors	4 Processors	8 Processors
35	27.08	46.38	70.94	88.07
34	25.57	44.69	68.83	87.00
33	23.77	43.24	68.47	86.98
32	22.46	41.78	67.05	86.27
31	21.08	40.42	66.13	85.90
30	19.81	39.07	65.07	85.43

Table 13: Load Sharing Percentage among different subtasks for c-fat200-2 dataset

	Load Sharing Percentage of POP-MBC			
	Without round robin strategy		With round robin strategy	
Size	4 Processors	8 Processors	4 Processors	8 Processors
5	41.71	47.24	60.37	72.91
6	40.78	46.45	60.07	73.48
7	40.81	46.94	60.1	73.10
8	43.73	50.57	60.21	72.77
9	44.47	52.61	60.12	71.57
10	44.57	53.73	59.87	70.74

The space complexity of POP-MBC algorithm is *O(mn)* where *m* and *n* are number of row vertices and column vertices respectively. The POP-MBC algorithm explores the search space in depth first manner and at any time only a path of the tree is stored in memory. Also, bit vectors are used for representing the dataset in memory as well as for processing. Hence, POP-MBC algorithm is highly memory efficient.

Table 14: Pattern distribution among different subtasks for c-fat200-1 dataset

	Distribution of Maximal Biclique patterns among Different Subtasks			
support	Subtask 1	Subtask 2	Subtask 3	Subtask 4
	Without Round Robin Strategy			
1	2583	56905	1159	19043
2	2533	56765	1109	18883
3	2280	55420	915	17607
4	1706	49794	583	13319
5	956	35824	198	5094
	With Round Robin Strategy			
1	9542	31090	18415	20643
2	9492	30992	18317	20489
3	9185	30171	17539	19327
4	8069	26967	14868	15498
5	5530	19218	9266	8058

Table 15: Load Sharing Percentage for c-fat500-2 dataset

	Load Sharing Percentage of POP-MBC			
	Without round robin strategy		With round robin strategy	
Supp-ort	4 Processors	8 Processors	4 Processors	8 Processors
5	50.32	43.39	62.25	73.44
6	51.07	43.57	61.92	73.67
7	49.49	42.59	61.77	73.48
8	47.07	41.06	61.27	73.77
9	44.26	39.18	60.05	74.05
10	41.64	37.28	58.37	74.26

Table 16: Load Sharing Percentage for johnson16-2-4 dataset

	Load Sharing Percentage of POP-MBC	
	Without round robin strategy	With round robin strategy
support	4 processors	4 Processors
1	22.61	54.76
2	22.18	53.75
3	22.2	53.12
4	21.06	52.67
5	20.42	51.29
6	19.83	50.9

Table 17: Pattern distribution among different subtasks for johnson16-2-4 dataset

support	Distribution of Maximal Bicliques among Different Subtasks			
	Subtask 1	Subtask 2	Subtask 3	Subtask 4
	Without Round Robin Strategy			
1	250	1817	10206	53229
2	220	1787	10176	53079
3	220	1787	10176	53079
4	163	1669	9997	52313
5	163	1669	9997	52313
	With Round Robin Strategy			
1	6445	17380	4047	37630
2	6414	17336	4017	37495
3	6414	17336	4017	37495
4	6297	17083	3915	36847
5	6297	17083	3915	36847

Table 18: Peak main memory usage in bytes for johnson16-2-4 dataset.

Supp-ort	Running time in Seconds and Peak Memory Usage in Bytes			
	LCM-MBC		POP-MBC	
	Running Time	Peak Memory Usage	Running Time	Peak Memory Usage
1	5.125	811008	3.25	569344
2	5.109	811008	3.281	569344
3	5.125	811008	3.391	569344
4	5.125	811008	3.516	569344
5	5.11	811008	3.703	569344
6	5.109	811008	3.922	569344

5. CONCLUSION

Efficient mining of maximal bicliques from graph datasets is a fundamental step to several data mining applications and we have proposed a fast and memory efficient parallel algorithm in this paper. The POP-MBC algorithm adaptively creates subtasks using round robin strategy which achieves very high load sharing among different processors. The POP-MBC algorithm is highly memory efficient because it does not store the previously computed patterns in main memory, the search space is explored in depth first manner and bit vectors are used for processing. The POP-MBC algorithm is guaranteed to complete its execution as long as the input dataset fits into the main memory and any number of processors can be used. Currently, we are investigating more optimizations to further reduce the overall running of the algorithm at each of the subtasks.

ACKNOWLEDGEMENTS

We wish to thank the authors of DCI_Closed algorithm and LCM-MBC algorithm for responding to our queries. We thank the people supporting DIMACS website for providing the synthetic dataset generator and benchmark graph datasets.

6. REFERENCES

[1] Jinyan Li, Guimei Liu, Haiquan Li, Limsoon Wong. 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,* Dec. 2007, Vol. 19, No. 12, pp. 1625-1637.

[2] Liping Ji, Kian-Lee Tan, K. H. Tung. Compressed hierarchical mining of frequent closed patterns from dense data sets. *IEEE Trans. on Knowledge and Data Engineering,* Sept 2007, Vol 19, No. 9.

[3] C. Lucchese, S. Orlando and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering,* January 2006, Vol. 18, No. 1, p. 21-36.

[4] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P.L. Hammer and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics,* 2004, 145(1), pp. 11-21.

[5] René Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics,* 2003, 131(3), pp. 651-654.

[6] V.M. Dias, C.M. de Figueiredo, and J. L. Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Journal of Theoretical Computer Science,* 2005, Vol. 337, pp. 240-248.

[7] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques, *in Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, Springer-Verlag, 2004, pp. 260–272.

[8] D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters,* 1994, Vol. 51, pp. 207-211.

[9] Mingjun Song, Sanguthevar Rajasekaran. A transaction mapping algorithm for frequent itemsets mining. *IEEE Transactions on Knowledge and Data Engineering,* April 2006, Vol. 18, No. 4, p. 472-481.

[10] G. Grahne, J. Zhu. Fast algorithms for frequent itemset mining using FP-trees. *IEEE Transactions on Knowledge and Data*

Engineering, October 2005, Vol. 17, No. 10, p. 1347-1362.

[11] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, Y. Yiu. MAFIA: a maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering,* November 2005, Vol. 17, No. 11, p. 1490-1504.

[12] S. Selvan and R. V. Nataraj. Efficient mining of maximal patterns using order preserving generators, *in proc. 16th Intl. Conf. on Advanced Computing and Communications,* Chennai, India, Dec. 2008.

[13] J. Besson, C. Robardet, J.F. Boulicaut, S. Rome. Constraint based concept mining and its application to microarray data analysis. *Journal of Intelligent Data Analysis*, 2005, pp. 59-82.

[14] Ji Liping. Mining localized co-expressed gene patterns from microarray data. *PhD dissertation,* School of Computing, National University of Singapore, June 2006.

[15] Ji Liping, K.L. Tan and A.K.H. Tung. Mining frequent closed cubes in 3D datasets. *Proc. 32nd int. conference on very large data-bases,* 2006.

[16] Gao Cong, Kian-Lee Tan, A.K.H. Tung, Feng Pan. Mining frequent closed patterns in microarray data, *ICDM'04,* Nov 2004, Vol. 1, Issue 4, p. 363-366.

[17] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining frequent pattern without candidate generation: a frequent pattern approach. *Journal of Data Mining and Knowledge Discovery,* 2004, Springer, p. 53-87.

[18] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases, *in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data,* Washington, DC, May 1993, p. 207-216.

[19] R. Agrawal and R. Srikant. Fast algorithms for mining association rules, *in Proceeding of Int. Conf. Very Large Data Bases,* Santiago, Chile, September 1994, p. 487-499.

[20] S. Brin, R. Motwani, J.D. Ullman and S. Tsur. Dynamic itemset counting and implication rules for market basket data, *SIGMOD Record,* June 1997, 6(2), p. 255-264.

[21] R. Agrawal and J.C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering,* December 1996, Vol. 8, No. 6, p. 962-969.

[22] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules, *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, January 1999, p. 398-416.

[23] J. Pei, J. Han, and R. Mao. CLOSET: an efficient algorithm for mining frequent closed itemsets, *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00),* May 2000, p. 11-20.

[24] J. Wang, J. Han and J. Pei. CLOSET+: searching for the best strategies for mining frequent closed itemsets, *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining,* August 2003, p. 236-245.

[25] M.J. Zaki, C.J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Trans. on Knowledge and data Engineering,* April 2005, Vol. 17, No. 4, p. 462-478.

[26] Ahmed Shakil, Frans Coenen, Paul Leng. Tree based partitioning of data for association rule mining. *Knowledge and Information Systems,* Springer, October 2006, Vol. 10, No. 3, p. 315-331.

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

[28] Yew-kwong Woon, Wee-keong Ng, Ee-Peng Lim. A support-ordered tree for fast frequent itemset discovery. *IEEE Transactions on Knowledge and Data Engineering,* July 2004, Vol. 16, No. 7, p. 875-879.

[29] H.D.K. Moonesinghe, Samah Fodeh, P.N. Tan. Frequent closed itemset mining using prefix graphs with an efficient flow-based pruning strategy. *ICDM'06.*

[30] Guimei Liu. Supporting efficient and scalable frequent pattern mining. *PhD Thesis,* Hong Kong University, May 2005.

[31] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *Bioinformatics,* 1998, Vol. 14, p. 55-67.

[32] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns, *KDD'04,* Seattle, Washington, August 2004.

R.V. Nataraj, *received the bachelor degree in computer science and engineering from Bharatiyar University, Coimbatore, India, in 2002 and Master degree in computer science from SASTRA University, Tanjore, India in 2004. He is currently* *pursuing PhD in Computer Science at PSG College of Technology, Anna University. He is a member of IEEE.*

His research interests include data mining, graph theory, combinatorial optimization and high performance computing.

***S.Selvan** received the B.E. degree in electronics and communication engineering and the M.E. degree in communication systems from the University of Madras, Chennai, India, in 1977 and 1979, respectively, and the Ph.D. degree in computer science and engineering from the Madurai Kamaraj University, Madurai, India, in 2001. He has 30 years of teaching and research experience. He is currently working as Principal and Professor of computer science and engineering at St. Peter's engineering college, Chennai, India. He is a senior member of IEEE and fellow of IE(I) and IETE. He has published more than 150 papers in international and national journals and conference proceedings.*

His areas of research include data mining, soft computing, computer networks, signal processing, image processing and network security.