Concise Papers

Efficient Mining of Large Maximal Bicliques from 3D Symmetric Adjacency Matrix

S. Selvan, *Senior Member*, *IEEE*, and
R.V. Nataraj, *Member*, *IEEE*

Abstract—In this paper, we address the problem of mining large maximal bicliques from a three-dimensional Boolean symmetric adjacency matrix. We propose CubeMiner-MBC algorithm which enumerates all the maximal bicliques satisfying the user-specified size constraints. Our algorithm enumerates all bicliques with less memory in depth first manner and does not store the previously computed patterns in the main memory for duplicate detection. To efficiently prune duplicate patterns, we have proposed a subtree pruning technique which reduces the total number of nodes that are processed and also reduces the total number of duplicate patterns that are generated. We have also incorporated several optimizations for efficient cutter generation and closure checking. Experiments involving several synthetic data sets show that our algorithm takes less running time than CubeMiner algorithm.

Index Terms—Data mining, maximal bicliques, algorithms, mining methods.

1 Introduction

Enumeration of maximal bicliques (also known as complete bipartite subgraph) can model several applications in the field of data mining including web mining, bioinformatics, and telecommunication usage data analysis [1]. However, the maximal biclique mining is a computationally demanding task [8] and the running time of the algorithm increases exponentially with respect to the number of vertices of the given graph. Several algorithms have been proposed for maximal biclique mining including Eppstein algorithm [11] and MICA [7]. Recently, the relationship between closed item set mining and maximal biclique has been well studied in [1] and LCM-MBC algorithm has been proposed for maximal biclique mining which extends the LCM closed item set mining algorithm [12] to generate maximal bicliques. However, these maximal biclique mining algorithms are limited to 2D data sets. The 2D data sets can be extended to 3D data sets by adding another dimension and enumeration of maximal bicliques from a 3D data set gives more useful information. One typical example is the web network data. Web communities are discovered by identifying maximal bicliques from web networks [1] and adding a dimension such as month/week gives rise to 3D data. In 2D version, the users are represented by vertices and their interactions are represented as edges [1]. Such a two-dimensional model of the scenario is not efficient as it does not focus on the strength of interaction. Also, if a weighted graph is used to counter this problem, it does not account for the pattern of the interaction with time. Therefore, if this is added as the third dimension (maybe in slices of month or year, as required), it will be useful in analyzing the density of interaction over a period. Thus, three-dimensional modeling of the interaction using 3D adjacency matrix provides more information and biclique patterns from 3D adjacency matrix represent the interactions as well as their strength over time. The same idea can be extended to mobile communication networks to discover interacting customer communities [1]. Like LCM-MBC algorithm, which extends the LCM algorithm, it is possible to use the existing 3D closed pattern mining algorithms such as CubeMiner [2], [6] and TRIAS [17] for 3D maximal biclique enumeration. However, this is not computationally efficient since all the maximal biclique patterns will be generated twice [1]. The symmetry property of the graph data set can be exploited to develop efficient algorithms for 3D maximal bicliques mining. In this paper, we propose CubeMiner-MBC algorithm for efficient mining of 3D maximal bicliques from 3D Boolean adjacency matrix containing no self loops. CubeMiner-MBC algorithm applies a subtree pruning technique, derived from the symmetric property of the graph data set, which prunes certain nodes in the enumeration tree and reduces the overall running time of the algorithm. The CubeMiner-MBC algorithm also incorporates several optimizations for efficient cutter generation and closure checking.

The rest of the paper is organized as follows: Section 2 presents the preliminaries associated with this paper. In Section 3, we present the subtree pruning technique, CubeMiner-MBC algorithm and its description. Section 4 analyzes the experimental results comprehensively while Section 5 concludes the paper.

2 Preliminaries

In this section, we present the basic definitions that are associated with this paper followed by the problem definition. Let $\mathcal{D} = \{\mathcal{H}, \mathcal{R}, \mathcal{C}\}$ be the 3D data set and let $\mathcal{H} = \{H_1, H_2, \ldots H_n\}$ be the set of adjacency matrices. Let $\mathcal{R}_i = \{R_1, R_2, \ldots R_m\}$ be the set of row vertices and $\mathcal{C}_i = \{C_1, C_2, \ldots C_m\}$ be the set of column vertices of the adjacency matrix H_i where $1 \leq i \leq n$. Throughout this paper, we assume that H_i represents an undirected graph without self loops. A subgraph $P_i = \{R : C\}$ is a complete bipartite subgraph of H_i iff $R \subseteq \mathcal{R}_i$, $C \subseteq \mathcal{C}_i$, and all the row vertices interact with all the column vertices. P_i is a maximal complete bipartite subgraph iff $\neg\exists\ \mathrm{r} \in \{\mathcal{R}_i \backslash R\}$ such that $(r, C) = 1$ (i.e., there is an edge between each c in C and r) and $\neg\exists\ \mathrm{c} \in \{\mathcal{C}_i \backslash C\}$ such that $(R, c) = 1$. In the context of closed pattern mining, P_i is a maximal bipartite subgraph iff $R : C$ forms a closed pattern in H_i. An example for 3D Boolean adjacency matrix with three height instances is given in Table 1. A pattern $P = \{H : R : C\}$ is a 3D maximal biclique iff $\neg\exists\ \mathrm{h} \in \{\mathcal{H} \backslash H\}$ such that $\mathrm{hx(RxC)} = 1, \neg\exists\ c \in \{\mathcal{C} \backslash C\}$ such that $\mathrm{cx}(H\mathrm{x}R) = 1$ and $\neg\exists\ \mathrm{r} \in \{\mathcal{R} \backslash R\}$ such that $\mathrm{rx(HxC)} = 1$. In other words, $H : R : C$ is a 3D maximal biclique iff $H : R : C$ forms a closed pattern in $\mathcal{D}$. A 3D maximal biclique is said to be of large size with respect to the user-specified height size constraint (min_h), column size constraint (min_c), and row size constraint (min_r) iff $|H| \geq min\_h, |R| \geq min\_r$, and $|C| \geq min\_c$. For example, $H_1\ H_2\ H_3 : R_3\ R_4 : C_1\ C_5$ is a large 3D maximal biclique subgraph for the data set given in Table 1 with respect to $min\_r = 2, min\_c = 2$, and $min\_h = 2$ constraints.

Problem definition. Given a 3D symmetric adjacency matrix, the problem is to mine all the large maximal bicliques in 3D context satisfying the user-specified size constraints (min_h, min_r, and min_c).

Since CubeMiner-MBC extends CubeMiner algorithm [2], we provide a short description of CubeMiner. The CubeMiner algorithm processes the entire data set as a whole for generating

- *S. Selvan is with the Department of Computer Science, Francis Xavier Engineering College, 103 G2, By Pass Road, Vannarapettai, Tirunelveli 627003, Tamil Nadu, India. E-mail: drselvan@ieee.org.*
- *R.V. Nataraj is with the Department of Information Technology, PSG College of Technology, Peelamedu, Coimbatore 641004, Tamil Nadu, India. E-mail: rvn@ieee.org, rvnataraj@mail.psgtech.ac.in.*

Manuscript received 8 Mar. 2009; revised 8 Aug. 2009; accepted 11 Oct. 2009; published online 1 June 2010.
Recommended for acceptance by Z.-H. Zhou.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-03-0121.
Digital Object Identifier no. 10.1109/TKDE.2010.97.

TABLE 1
An Example for 3D Adjacency Matrix

H_1

	C_1	C_2	C_3	C_4	C_5
R_1	0	1	1	1	1
R_2	1	0	1	0	0
R_3	1	1	0	0	1
R_4	1	0	0	0	1
R_5	1	0	1	1	0

H_2

	C_1	C_2	C_3	C_4	C_5
R_1	0	1	1	1	0
R_2	1	0	1	1	0
R_3	1	1	0	0	1
R_4	1	1	0	0	1
R_5	0	0	1	1	0

H_3

	C_1	C_2	C_3	C_4	C_5
R_1	0	1	1	1	0
R_2	1	0	1	0	0
R_3	1	1	0	1	1
R_4	1	0	1	0	1
R_5	0	0	1	1	0

3D patterns. The algorithm generates a ternary tree and visits the ternary tree in depth first manner. The root node contains the entire height set, row and column vertex set and the algorithm uses cutters for generating the left, middle, and right child of a node. A cutter is a subcube, $h' \times r' \times C'$ such that none of their elements are in relation with others, i.e., in the context of Boolean adjacency matrix they contain false ("0") values. Since h' and r' contain only one element each, they are denoted using lower case letters. The cutters are recursively applied and the child nodes will have less number of false values than their parents. If $H : R : C$ is the node, then the left son, middle son, and right son are generated as follows: $LS = \{H \backslash h' : R : C\}$, $MS = \{H : R \backslash r' : C\}$, and $RS = \{H : R : C \backslash C'\}$. The leaf nodes contain no false values and all the leaf nodes are 3D closed patterns. For the example data set given in Table 1, the root node is $\{(H_1\ H_2\ H_3), (R_1, R_2, R_3, R_4, R_5), (C_1, C_2, C_3, C_4, C_5)\}$. The cutter for the root node is $\{H_1 : R_1 : C_1\}$. The left son, middle son, and right son are $\{(H_2\ H_3), (R_1, R_2, R_3, R_4, R_5), (C_1, C_2, C_3, C_4, C_5)\}$, $\{(H_1\ H_2\ H_3), (R_2, R_3, R_4, R_5), (C_1, C_2, C_3, C_4, C_5)\}$ and $\{(H_1\ H_2\ H_3), (R_1, R_2, R_3, R_4, R_5), (C_2, C_3, C_4, C_5)\}$, respectively. While generating every node, several checks are performed to ensure their unicity and closeness. The following checks are performed while generating the left son: size constraint check, left track check, and close row set check. While generating the middle son, the size constraint check, middle track check, and close height set check are performed. For generating right son, the following checks are performed: size constraint check, close row set check, and close height set check. Note that middle track and left track checking are done to ensure the unicity. Left track checking ensures that the height atom of the cutter is never removed in the middle and right son subtree whereas middle track checking ensures that the row atom of the cutter is never removed in the right son subtree. For more details, readers may refer [2].

3 3D Maximal Biclique Mining

The existing CubeMiner algorithm can be directly applied for generating 3D biclique patterns from symmetric adjacency matrices. However, there is a major disadvantage with this approach, i.e., all the 3D maximal biclique patterns will be generated twice [1]. We can avoid printing duplicate biclique patterns by comparing the minimum element of row vertices and the minimum element of column vertices, i.e., if *min(row vertex set)* is lexicographically greater than *min(column vertex set)*, then the pattern can be outputted. It is easy to verify the correctness of this technique. For example, let $H : R : C$ be a 3D pattern and $H' : R' : C'$ be another pattern which is a duplicate. If $min(R) > min(C)$, then $min(R') < min(C')$. Hence, only one pattern would be printed. Notice that, if a vertex is present in R then the same vertex will not be present in C since the data set is assumed to contain no self loops. (It is to be noted that, if self loops are allowed then the number of closed patterns need not be even and Lemma 1 stated in this paper may not hold true. The property of self loops and its relation to closed pattern mining is well discussed in [1].) The same technique has been adopted in [1] for the removal of duplicate patterns. Even if we use this technique to avoid outputting duplicate patterns, the problem is that we generate all the duplicate patterns and this is a computationally expensive task and the algorithm takes more running time. Is there any way to avoid the generation of duplicate patterns? We provide an answer to this question using CubeMiner-MBC algorithm. The CubeMiner-MBC algorithm extends the CubeMiner algorithm by including a subtree pruning strategy which avoids the generation of duplicate patterns to some extent (for dense data sets, duplicate patterns are avoided to a greater extent). The quantified results related to duplicate elimination are discussed in Section 4. The CubeMiner-MBC algorithm also includes an efficient cutter generation strategy using row cutter index and height cutter index which is later explained in this section. Also, an efficient closeness checking scheme is proposed which reduces the running time of the algorithm for dense data sets.

3.1 Subtree Pruning

The CubeMiner algorithm generates a ternary tree with patterns in the leaf node. From our detailed analysis, we have concluded that the right subtree of the root node need not be generated for the root node since all the patterns of the right subtree are duplicates with respect to some of the patterns of the middle son subtree of the root node, i.e., the symmetric pair of the patterns generated in the right subtree will be generated in the middle son subtree. Also, all the right son subtree of the left node with no middle son or right son in their path from the root can be pruned since the symmetric pair of the patterns that are generated in the right subtree will be generated in their corresponding middle son subtree.

We define the symmetric pair of a 3D biclique pattern as follows: Let H : R : C be a 3D biclique pattern. Then, its symmetric pair is H : C : R. For example, if $H_1\ H_2\ H_3 : R_3\ R_4 : C_1\ C_5$ is a pattern, then its symmetric pair is $H_1\ H_2\ H_3 : R_1\ R_5 : C_3\ C_4$. We denote a pattern as P and its symmetric pair pattern as P'.

Lemma 1. *Let $\mathcal{D}$ be the data set and $\mathcal{P}$ be the set of maximal biclique subgraph patterns of $\mathcal{D}$. Then, $\forall P \in \mathcal{P}$, $P' \in \mathcal{P}$.*

Proof. Refer [1]. □

Lemma 2. *Let Z be the first cutter applied at the root node. Let h' be the height atom of Z, r' be the row atom of Z and C' be the column atom of Z. Then, all patterns that include h' will be generated only in the subtree of middle son and right son of the root node. Also, all the patterns that are generated will include h'.*

Proof. Since h' is removed from the left son node, the subtree of the left node will not generate any pattern that includes h'. Hence, patterns that include h' will only be generated in the right son and middle son subtree. Also, the left track checking ensures that the h' is never removed in the middle son and

right son subtree. Hence, all the patterns that are generated will include h'. □

Lemma 3. *Let Z be the cutter of the root node with h', r', and C' be their height, row, and column atoms. All patterns that include h' and r' will be generated only in the right son subtree of the root node.*

Proof. According to Lemma 2, all the patterns will include h'. According to middle track checking, r' will never be removed in the right son subtree whereas r' is removed in the middle son subtree. Hence, all patterns that include h' and r' will only be generated in the right son subtree. □

Lemma 4. *Let P be a maximal biclique and P′ be the symmetric pair of P. If P is generated in the right son subtree of the root node then P′ will be generated only in the middle son subtree of the root node.*

Proof. Let $\mathcal{H}$:$\mathcal{R}$:$\mathcal{C}$ be the root node. Let Z be the cutter applied at the root node. Let h', r', and C' be the height, row, and column atoms of Z. Since the graph data set is assumed to be containing no self loops, the following holds: $\mathrm{C}' \supseteq \mathrm{r}'$. By applying a cutter Z at the root node, we get a left son (LS), a middle son (MS), and a right son (RS) as follows: $\mathrm{LS} = \{\mathcal{H}\backslash h' : \mathcal{R} : \mathcal{C}\}$, $\mathrm{MS} = \{\mathcal{H} : \mathcal{R}\backslash r' : \mathcal{C}\}$, and $\mathrm{RS} = \{\mathcal{H} : \mathcal{R} : \mathcal{C}\backslash C'\}$. All the patterns, $P' = H'$:R':C', in the right subtree include r' in R' (Lemmas 2 and 3). Hence, by symmetric property, P should contain r' in C'. But in the right son, r' is removed from its column set and hence the symmetric pair will not occur in the right son subtree. Hence, if P' occurs in the right son subtree of the root node then P occurs in the middle son subtree of the root node. □

Example. Let us consider the root node and its cutter for the example data set given in Table 1. The root node is $\{(H_1\ H_2\ H_3), (R_1,\ R_2,\ R_3, R_4,\ R_5), (C_1,\ C_2,\ C_3,\ C_4,\ C_5)\}$ and the cutter is $H_1 : R_1 : C_1$. According to Lemmas 2, 3, and 4, all the patterns of the right subtree will not include C_1 whereas R_1 is included in all the patterns. Hence, symmetric pair of the right son subtree must include C_1 in their column and patterns with C_1 in their column will be generated only in the middle son subtree.

Lemma 5. *Let $\mathcal{LS}$ be the set of Left son nodes with no middle son or right son in their path from the root node. Then, $\forall\ LS \in \mathcal{LS}$, right son of LS can be pruned.*

Proof. Let Z be the cutter applied at the root node with h' as height atom. It should be noted that, in the left son of the root node, a height element, h', is removed and the resulting datum is another 3D data set (provided there are at least two elements in the height set). Hence, according to Lemma 4, the right subtree of the left son can be pruned. □

It should be noted that, this subtree pruning technique can be applied only when $min\_r = min\_c$ because all the patterns will be generated twice. If $min\_r\ != min\_c$, the algorithm may prune one of the patterns as not satisfying the size constraint. For example, consider the following pattern $P = H_1\ H_2\ H_3 : R_1 : C_2\ C_3\ C_4$. The symmetric pair, P', of the pattern P is $H_1\ H_2\ H_3 : R_2\ R_3\ R_4 : C_1$. If $min\_r = 1$ and $min\_c = 3$ then P will be generated whereas P' will not be generated since P' does not satisfy the min_c size constraint.

3.2 CubeMiner-MBC Pseudocode

INPUT: $\mathcal{R}$ (set of row vertices), $\mathcal{C}$ (set of column vertices), $\mathcal{H}$ (set of heights) and size constraints (min_r, min_c, min_h)

OUTPUT: set of maximal bicliques satisfying the size constraints.

1. *construct the data set D in memory after removing the rows, columns that are not supported by min_r and min_c constraint from all the height sets*
2. *$h' = r' = C' = null$ (cutter atoms are initialized to null)*
3. *$MC = LC = null$ (meant for left and middle track checking)*
4. *Call CubeMiner-MBC($\mathcal{H}, \mathcal{R}, \mathcal{C}$)*

5. *CubeMiner-MBC(H, R, C)*
6. *{*
7. *while(true)*
8. *generate cutter for the current node*
9. *if (cutter-exists)*
10. *update h', r', C', LC*
    //generate right child
11. *if (!lemma 5) **//subtree pruning***
12. *if $|C\backslash C'| \geq min\_c$*
13. *push H, R, $C\backslash C', MC \cup r'$, LC to stack*
14. *endif*
15. *endif*
    //generate middle child
16. *if $|R\backslash r'| \geq min\_r$*
17. *if $\exists\ r' \in$ MC **//middle track check***
18. *discard middle child(unicity constraint)*
19. *else*
20. *push H, $R\backslash r'$, C, MC, LC to stack*
21. *endif*
22. *endif*
    //generate left child
23. *if $|H\backslash h'| \geq min\_h$*
24. *if $h'! = LC$ **//left track check***
25. *$H = H\backslash h'$*
26. *continue*
27. *endif*
28. *endif*
29. *else **//cutter does not exist***
    //check for height closure and row closure
30. *if $\neg\exists\ h \in (\mathcal{H}\backslash H)\&\&(R \times C) \in h$*
31. *if $\neg\exists\ r \in (\mathcal{R}\backslash R)\&\&(H \times C) \in r$*
32. *if min(R) > min(C)*
33. *write H : R : C as 3D maximal biclique*
34. *endif*
35. *endif*
36. *endif*
37. *endif*
38. *if (stack not empty)*
39. *pop from stack to H, R, C MC, LC*
40. *continue*
41. *else*
42. *break*
43. *endif*
44. *endwhile*
45. *}*

3.3 Description

The CubeMiner-MBC algorithm starts with a root node containing the entire height set, row set vertices, and column set vertices. For the root node, the algorithm generates a cutter (line no. 8) and pushes the right son (line no. 13), and middle son (line no. 20) into the stack. While pushing the right son, the algorithm checks whether it satisfies the Lemma 5 constraint (line no. 11) and size constraint (line no. 12). Similarly, while pushing the middle son, the algorithm checks for the size constraint on row vertex set (line no. 16) and middle track constraint (line no. 17). After pushing the middle and right son

TABLE 2
Data Sets Used

Dataset	Number of heights	Number of vertices	Average number of edges per vertex
Dataset-1	9	200	7 to 13
Dataset-2	5	500	12 to 16
Dataset-3	9	200	55 to 60
Dataset-4	5	500	80 to 85

into the stack, the algorithm processes the left son (provided left track constraint and size constraint are satisfied) and the process gets repeated. In the pseudocode, the variables H, R, and C are used to store the currently processed height set, row vertex set, and column vertex set, respectively. The h', r', and C' store the height atom, row atom, and column atoms of the current cutter. The variable LC stores the previously applied height atom for left track checking and the variable MC is used to store the row atom of the right son cutter for middle track checking. For the proof of left track checking and middle track checking, readers may refer [2]. If no cutter exists, the algorithm checks for its height set closure (line no. 30) and row vertex set closure (line no. 31) of the 3D pattern. If the pattern is found to be closed, duplicate checking is done by comparing the minimum of row vertex set and column vertex set. It is to be noted that we can only compare whether *min(row) > min(column)* and we cannot compare whether *min(row) < min(column).* This constraint is due to the subtree pruning strategy. Note that the min element of row vertex set is always present in all the patterns that are generated in the pruned right subtree. Hence, if we compare using "<" constraint, we would be missing the symmetric pair of the pruned patterns. Once the stack becomes empty, the algorithm terminates.

3.4 Optimizations

The CubeMiner-MBC algorithm generates a ternary tree and applies a subtree pruning technique which prunes branches that generate duplicate patterns. The cutter generation and closure checking are the major operations in the algorithm. The CubeMiner algorithm performs a closeness check whenever a node is generated, i.e., for the left son it performs row set closeness check, for a middle son it performs height set closeness check, and for a right son it performs both row set closeness and height set closeness check. The closeness checking is a computationally expensive task and hence we perform the closeness checking only in the leaf node (line nos. 30 and 31). Though this technique generates more number of nodes, our empirical results have shown that this technique has reduced the overall running time of the algorithm. The next major operation is the cutter generation since for every node that is created, a cutter is generated to build its left, right, and middle son. A simple approach is to generate all cutters, store them, and use these cutters one by one. However, this approach requires extra memory. In the CubeMiner-MBC algorithm, we generate cutters on the fly in an efficient manner using height cutter index and row cutter index. The height cutter index indicates the height element to be scanned and the row cutter index indicates the row to be processed. For the sake of simplicity, we have not shown the complete usage of the height cutter index and row cutter index in the algorithm. Our empirical results have shown that this technique has reduced the running time to a great extent.

3.5 Parallelization

The CubeMiner-MBC algorithm can be easily parallelized since the nodes can be processed independently and concurrently on several processors. The only requirement is the availability of the entire data set in all the processing nodes. Fortunately, distributing the data set to different processors is computationally negligible when compared to the mining task. However, there is a major issue to be addressed regarding load sharing across different processors. Our own empirical results have shown that the left subtree of the root node generates more number of patterns and takes more running time than the middle and right subtree. Hence, allocating nodes as such to all the processors will lead to poor load sharing among the processors. An efficient approach would be to generate as much number of nodes in the master processor in breadth first manner and assign these nodes to the slave processors. When a slave processor completes its execution, another node from the master processor can be allocated to this slave processor for processing. In this way, the overall running time can be reduced and the load sharing among different processors can be improved.

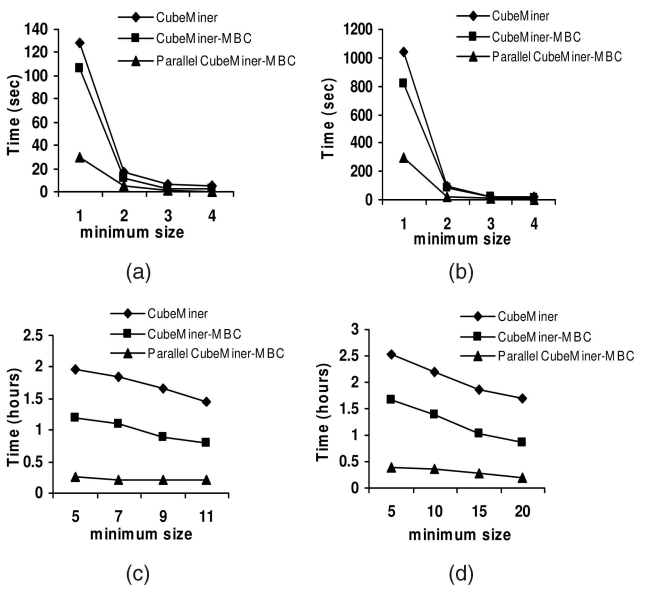

Fig. 1. Running time versus minimum size constraint for various synthetic data sets (min_h is assumed as 1). (a) Data set 1. (b) Data set 2. (c) Data set 3. (d) Data set 4.

4 Experimental Results

We have implemented the CubeMiner and CubeMiner-MBC algorithms using C language and the code was compiled using 32-bit Microsoft Visual C++ compiler. To our knowledge, we could not find a similar algorithm in the literature for mining maximal biclique patterns from 3D symmetric adjacency matrix. Hence, to make the comparison fair, we have included the optimizations discussed in Section 3.4 in the implementation of both the algorithms and the results mainly focus on the subtree pruning strategy which is the core contribution of this paper. All the experiments were conducted on Pentium 4 machines with 1 GB of main memory loaded with Windows XP operating system. We have created several synthetic graph data sets and the description of the data sets used in our experiments is given in Table 2. We have taken two sparse data sets and two comparatively dense data sets to clearly illustrate the effectiveness of the subtree pruning. Fig. 1 shows the associated results and we have used four processors for parallel CubeMiner-MBC. As the values of minimum size constraint are increased, the running time of the algorithm gets reduced because more nodes are likely to be pruned as not satisfying the minimum size constraint. If the values of minimum size constraint are decreased, more nodes are likely to satisfy the size constraint and hence the running time is increased. For sparse data sets, more number of nodes is likely to be pruned than the dense data sets if minimum size constraint values are

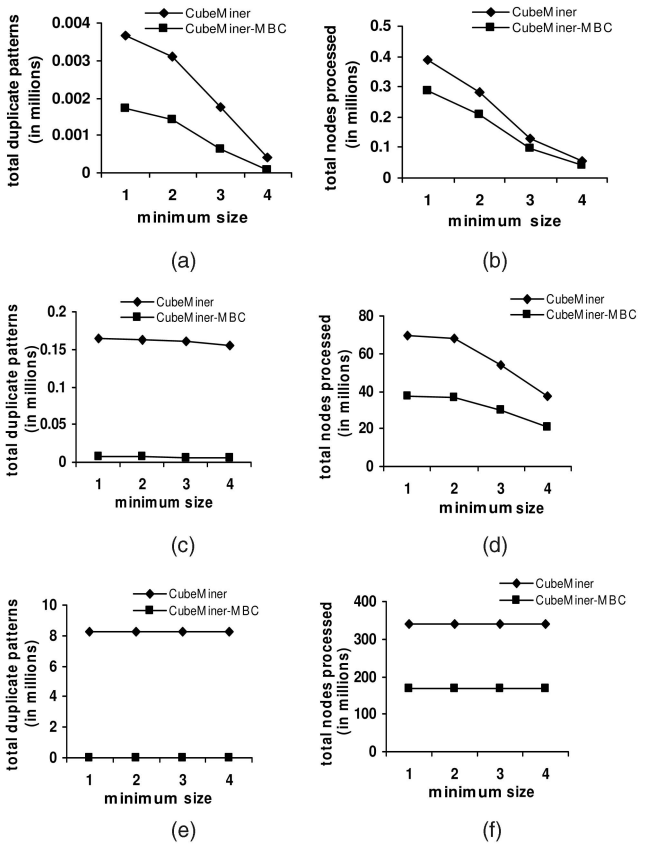

Fig. 2. Effectiveness of the subtree pruning strategy in terms of reduction in the total number of duplicate patterns generated and total number of nodes processed. (a) Data set 5. (b) Data set 5. (c) Data set 6. (d) Data set 6. (e) Data set 7. (f) Data set 7.

increased. Hence, the running time is reduced drastically for sparse data sets (Figs. 1a and 1b) when compared to the dense data sets (Figs. 1c and 1d). For sparse data sets, the subtree pruning strategy prunes less duplicate patterns and hence the difference in running time is comparatively low. The reason is that the pruned right son subtree contains less information because more number of column vertex elements is removed. For dense data sets, the pruned right son subtree contains more information because less number of column elements is removed in the right son and hence, more number of duplicate patterns is pruned. If only one column element is removed while generating the pruned right son nodes, then no duplicates will be generated (a result illustrating zero duplicate patterns is shown in Fig. 2e). The reason is that, for such cases, the child's submatrices are transposes of each other and hence the number of patterns contained in subtree of the child nodes is same with one set of patterns being duplicates. In other words, if we produce two submatrices from a symmetric matrix by removing a particular row and the corresponding column, then the resultant submatrices are transposes of each other.

TABLE 3
Data Sets to Analyze the Effectiveness of Subtree Pruning

Dataset	Number of heights	Number of vertices	Average number of edges per vertex
Dataset-5	5	30	12
Dataset-6	5	30	15
Dataset-7	5	30	25

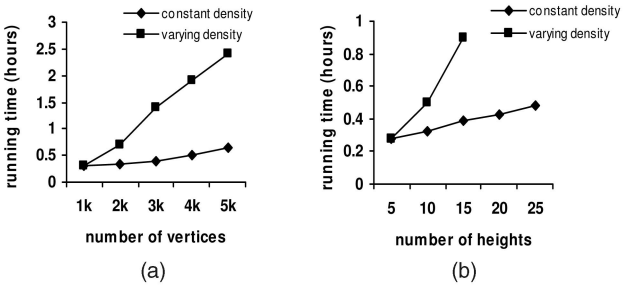

Fig. 3. Results on algorithm's scalability.

To demonstrate the effectiveness of the subtree pruning technique in terms of total duplicates pruned and total nodes processed, we have created four small data sets with varying density as shown in Table 3. We could not get the results in reasonable time for large dense data sets and hence we created these small data sets (for a particular dense data set with 500 vertices, the running time of the algorithm exceeded 24 hours and we terminated the execution). Table 3 shows the data set characteristics and the associated results are shown in Fig. 2. As shown in the results, the number of duplicate patterns generated is inversely proportional to the data set density, i.e., effectiveness of the pruning technique is very high for dense data sets. Also, the subtree pruning technique reduces the total number of nodes that are processed to a great extent for dense data sets.

Scalability. Both CubeMiner and CubeMiner-MBC algorithms are highly scalable. Note that, at any time, only a path of a tree and the data set is stored in the main memory. Hence, as long as the data set fits into the main memory, the algorithm is guaranteed to complete its execution. To assess the scalability, we have conducted several experiments and we present a summarized result in Fig. 3. The graph shown in Fig. 3 illustrates that the running time of the algorithm increases minimally as we increase the number of vertices (keeping the density constant) whereas the running time increases linearly if both density and the number of vertices (number of heights in Fig. 3b) are increased. For highly dense data sets, parallelized execution is the only way to get the results in reasonable amount of time and the algorithm can be easily parallelized to any number of processors as discussed in Section 3.5.

Related algorithm. The following analyzes a very recently proposed DataPeeler algorithm [3], since DataPeeler can also be used to mine 3D bicliques. Unlike CubeMiner, which uses ternary tree enumeration strategy to generate closed 3-sets, the DataPeeler algorithm uses a binary tree enumeration strategy to generate closed n-sets. Each node in the binary tree contains two n-sets (U & V). For the root node, U is set to null whereas V contains the entire n-sets. For a node, the child nodes are generated by choosing an element from V. If v is the chosen element, then the left child and right child are $\{U \cup v, V \backslash v\}$ and $\{U, V \backslash v\}$, respectively. While generating child nodes, the algorithm ensures that $U \cup v$ is connected. If $U \cup v$ is not connected, the algorithm chooses another element from V to continue the enumeration and v is discarded. Once the V-set is null, the algorithm performs closure checking of U-set using the elements that are removed from the set of right child nodes in its path. If the U-set is found to be closed, the algorithm outputs U as a closed n-set. The algorithm also uses several optimizations to speed up the running time. In a nutshell, the algorithm tries to grow the connected n-set (note that a maximal connected n-set is a closed n-set). For complete details, readers may refer [3].

If DataPeeler algorithm is applied as such on a 3D adjacency matrix to generate maximal bicliques, all the biclique patterns will be generated twice. Hence, to prune duplicate patterns, specialized

pruning techniques need to be developed for DataPeeler algorithm. We have not compared CubeMiner-MBC with DataPeeler because our experimental results mainly focus on the subtree pruning strategy and hence, it is unfair to compare these two algorithms. In our future work, we plan to develop pruning techniques for DataPeeler algorithm to prune duplicate patterns.

5 Conclusion

We have investigated the problem of mining maximal bicliques from 3D symmetric adjacency matrix and we have proposed a subtree pruning strategy which prunes certain nodes that generate only duplicate patterns. For dense data sets, our subtree pruning strategy reduces the total number of duplicate patterns to a great extent and for certain cases the CubeMiner-MBC algorithm generates all the 3D maximal biclique patterns with no duplicates. Our efficient implementation along with the subtree pruning and other optimizations, have reduced the overall running time of the algorithm. The algorithm is also highly memory-efficient since none of the results are stored in the main memory and requires only the data set and a path of the tree to be stored in the main memory. Though our algorithm generates the 3D maximal bicliques in an efficient manner, some research questions are still open. First, our subtree pruning strategy prunes only the right son subtree of the root node and some right son nodes in the left subtree of the root node. The research could be extended further to develop more pruning techniques in the middle son subtree by exploiting the symmetry property of the data set. Second, the algorithm could be extended to work on more than three dimensions and more subtree pruning techniques can be developed for high-dimensional data sets. Finally, massive parallel system infrastructure is required to get the results in reasonable time for large dense data sets.

Acknowledgments

The authors wish to thank the anonymous reviewers for their comments which helped them to enhance the paper. They also thank the authors of CubeMiner, D-Miner, and DataPeeler algorithms for responding to their queries.

References

[1] J. Li, G. Liu, H. Li, and L. Wong, "Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-One Correspondence and Mining Algorithms," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 12, pp. 1625-1637, Dec. 2007.

[2] J. Liping, K.L. Tan, and A.K.H. Tung, "Mining Frequent Closed Cubes in 3D Data Sets," *Proc. 32nd Int'l Conf. Very Large Data Bases*, 2006.

[3] L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut, "Closed Patterns Meet n-ary Relations," *ACM Trans. Knowledge Discovery from Data*, vol. 3, no. 1, pp. 1-36, 2009.

[4] L. Ji, K.-L. Tan, and K.H. Tung, "Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 9, pp. 1175-1187, Sept. 2007.

[5] J. Besson, C. Robardet, J.F. Boulicaut, and S. Rome, "Constraint Based Concept Mining and Its Application to Microarray Data Analysis," *Intelligent Data Analysis*, vol. 9, pp. 59-82, 2005.

[6] J. Liping, "Mining Localized Co-Expressed Gene Patterns from Microarray Data," PhD dissertation, School of Computing, Nat'l Univ. of Singapore, June 2006.

[7] 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 Math.*, vol. 145, no. 1, pp.11-21, 2004.

[8] R. Peeters, "The Maximum Edge Biclique Problem is NP-complete," *Discrete Applied Math.*, vol. 131, no. 3, pp. 651-654, 2003.

[9] V.M. Dias, C.M. de Figueiredo, and J.L. Szwarcfiter, "Generating Bicliques of a Graph in Lexicographic Order," *J. Theoretical Computer Science*, vol. 337, pp. 240-248, 2005.

[10] K. Makino and T. Uno, "New Algorithms for Enumerating all Maximal Cliques," *Proc. Ninth Scandinavian Workshop Algorithm Theory (SWAT '04)*, pp. 260-272, 2004.

[11] D. Eppstein, "Arboricity and Bipartite Subgraph Listing Algorithms," *Information Processing Letters*, vol. 51, pp. 207-211, 1994.

[12] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver.2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets," *Proc. Fourth IEEE Int'l Conf. Data Mining (ICDM '04) Workshop Frequent Itemset Mining Implementations (FIMI '04)*, 2004.

[13] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Pattern without Candidate Generation: A Frequent Pattern Tree Approach," *Data Mining and Knowledge Discovery*, vol. 8, pp. 53-87, 2004.

[14] M. Song and S. Rajasekaran, "A Transaction Mapping Algorithm for Frequent Itemsets Mining," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 4, pp. 472-481, Apr. 2006.

[15] G. Grahne and J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 10, pp. 1347-1362, Oct. 2005.

[16] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 1, pp. 21-36, Jan. 2006.

[17] R. Jaschke, A. Hotho, C. Schmitz, B. Ganter, and G. Stumme, "TRIAS: An Algorithm for Mining Iceberg Tri-Lattices," *Proc. Sixth IEEE Int'l Conf. Data Mining (ICDM '06)*, pp. 907-911, 2006.

[18] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. Int'l Conf. Very Large Data Bases*, pp. 487-499, Sept. 1994.

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

[20] M.J. Zaki and C.J. Hsiao, "Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 4, pp. 462-478, Apr. 2005.