

# Efficient Mining of Large Maximal Bicliques

Guimei Liu, Kelvin Sim, and Jinyan Li

Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613  
{visgmliu, shsim, jinyan}@i2r.a-star.edu.sg

**Abstract.** Many real world applications rely on the discovery of maximal biclique subgraphs (complete bipartite subgraphs). However, existing algorithms for enumerating maximal bicliques are not very efficient in practice. In this paper, we propose an efficient algorithm to mine large maximal biclique subgraphs from undirected graphs. Our algorithm uses a divide-and-conquer approach. It effectively uses the size constraints on both vertex sets to prune unpromising bicliques and to reduce the search space iteratively during the mining process. The time complexity of the proposed algorithm is  $O(nd \cdot N)$ , where  $n$  is the number of vertices,  $d$  is the maximal degree of the vertices and  $N$  is the number of maximal bicliques. Our performance study shows that the proposed algorithm outperforms previous work significantly.

## 1 Introduction

Graphs can be used to model a wide range of real world applications. In this paper, we study the problem of mining maximal complete bipartite subgraphs (not necessarily induced subgraphs) from undirected graphs. Complete bipartites are also called bicliques. A biclique has two disjoint sets of vertices, and there is an edge between two vertices if and only if the two vertices are in different vertex sets. A biclique is maximal if the biclique is not a proper subgraph of another biclique. Maximal bicliques have been used to solve the edge covering problem [7], and they have many other arising applications.

**Web community discovery.** Websites that are part of the same community frequently do not reference one another for many reasons [11]. Linkage between these related pages can nevertheless be established by a different phenomenon: related pages are frequently visited together. Related pages and the Web users visiting these pages form a biclique or a dense bipartite. Web communities can be discovered by first enumerating maximal bicliques from Web log data as community cores, and then find the rest of the community members using community cores.

**Topological structure discovery from protein-protein interaction networks.** In the last several years, high-throughput interaction detection approaches have led to the discovery of thousands of interactions between proteins. Some hidden topological structures discovered from protein-protein interaction networks, such as cliques and bicliques, consist of biologically relevant functional groups [6].

**Maximal concatenated phylogenetic dataset discovery.** A phylogenetic tree is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor. To improve the accuracy of tree reconstruction, phylogeneticists are extracting increasingly large multigene data sets from sequence databases [18]. Determining whether a database contains at least  $k$  genes sampled from at least  $m$  species is to determine whether there are  $k$  genes and  $m$  species forming a biclique. Complete phylogenetic datasets can be discovered by enumerating all the maximal bicliques satisfying the size constraints.

In real world problems, such as the applications described above, those bicliques with a very small vertex set are usually of no interest. It is therefore desirable to mine only large interesting bicliques. A biclique is large if the size of its both vertex sets is no less than a predefined threshold. Mining maximal bicliques has been studied previously. Alexe et al. [2] use consensus algorithms to enumerate all maximal bicliques, which may generate many uninteresting small bicliques. Li et al. [12] have proved that there is a correspondence between maximal bicliques and frequent closed itemsets. On one hand, a closed itemset and the set of transactions containing the closed itemset form a biclique. On the other hand, the adjacency matrix of a graph can be viewed as a transaction database, and a biclique corresponds to a pair of frequent closed itemsets in the transaction database. Li et al. suggest to use frequent closed itemset mining techniques [23,22,13,20] to mine maximal bicliques. However, the large maximal biclique mining problem has size constraints on both vertex sets, while the frequent itemset mining problem put size constraint on only one side—the transaction set. Traditional frequent itemset mining algorithms also use only the size constraint on transaction set to prune the search space. As a result, using frequent closed itemset mining algorithms to mine maximal bicliques also generates many small bicliques. Another problem with the frequent itemset mining approach is that each maximal biclique is generated twice in undirected graphs. This paper introduces the problem of mining large maximal biclique subgraphs and proposes an efficient algorithm to solve the problem. Our algorithm takes a divide-and-conquer approach, and it effectively uses the size constraints on both sides to iteratively prune the search space. Non-maximal bicliques as well as duplicate bicliques are pruned efficiently during the mining process.

The rest of the paper is organized as follows. Section 2 formulates the problem, our algorithm is described in Section 3. Section 4 reports experiment results. Related work is reviewed in Section 5. Section 6 concludes the paper.

## 2 Definitions and Properties

In this section, we formulate the problem of mining large maximal bicliques. An *undirected graph*  $G$  is defined as a pair  $(V, E)$ , where  $V$  is a set of vertices, and  $E$  is a set of edges between the vertices. Two vertices are *adjacent* if there is an edge between them. The *adjacency list* of a vertex  $v$  in  $G = (V, E)$ , denoted as  $\Gamma(v, G)$ , is defined as the set of vertices adjacent to  $v$ , that is,

$\Gamma(v, G) = \{u | \{u, v\} \in E\}$ . The adjacency list of a set of vertices  $X$  in  $G = (V, E)$  is defined as  $\Gamma(X, G) = \{u | u \in V \text{ and } \forall v \in X, \{u, v\} \in E\} = \{u | u \in V \text{ and } X \subseteq \Gamma(u)\}$ . We denote  $\Gamma(X, G)$  as  $\Gamma(X)$  if  $G$  is clear from the context. The adjacency lists of vertex sets have the **anti-monotone** property.

**Proposition 1.** *Let  $V_1$  and  $V_2$  be two sets of vertices in  $G = (V, E)$  and  $V_1 \subseteq V_2$ . We have  $\Gamma(V_2) \subseteq \Gamma(V_1)$ .*

Given a graph  $G = (V, E)$ , graph  $G' = (V', E')$  is a *subgraph* of  $G$  if  $V' \subseteq V$ ,  $E' \subseteq E$  and  $\forall \{u, v\} \in E'$ ,  $u, v \in V'$ . If  $V' \subset V$  or  $E' \subset E$ , then we say  $G'$  is a **proper subgraph** of  $G$ . A graph  $G = (V, E)$  is a *bipartite* if its vertex set  $V$  can be partitioned into two disjoint nonempty sets  $V_1$  and  $V_2$  such that every edge in  $E$  connects a vertex in  $V_1$  and a vertex in  $V_2$ , that is, no edge in  $E$  connects either two vertices in  $V_1$  or two vertices in  $V_2$ . Bipartite  $G$  is also denoted as  $G = (V_1, V_2, E)$ .

**Definition 1 (Biclique).** *A bipartite  $G = (V_1, V_2, E)$  is called a biclique if for each  $v_1 \in V_1$  and  $v_2 \in V_2$ , there is an edge between  $v_1$  and  $v_2$ , that is,  $E = \{\{u, v\} | u \in V_1, v \in V_2\}$ .*

The edge set  $E$  of a biclique  $G = (V_1, V_2, E)$  can be completely determined by the two vertex sets  $V_1$  and  $V_2$ , so we omit the edge set and denote a biclique  $G$  simply as  $G = (V_1, V_2)$ . Let  $G = (V, E)$  be an undirect graph,  $V_1$  and  $V_2$  be two subsets of  $V$ . If  $V_1, V_2$  and all the edges between  $V_1$  and  $V_2$  form a biclique subgraph of  $G$ , we say that  $V_1$  and  $V_2$  form a biclique subgraph of  $G$ . According to the definition, for any subset  $V_1$  of  $V$ ,  $V_1$  and  $\Gamma(V_1, G)$  form a biclique subgraph of  $G$ .

**Proposition 2.** *Let  $G' = (V_1, V_2, E')$  be a biclique subgraph of  $G = (V, E)$ . We have  $V_1 \subseteq \Gamma(V_2, G)$  and  $V_2 \subseteq \Gamma(V_1, G)$ .*

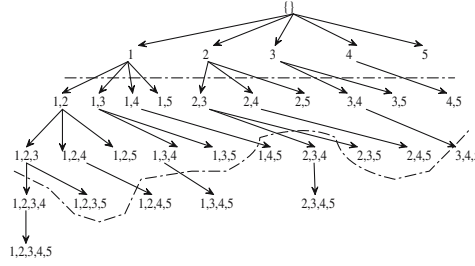
If  $G = (V_1, V_2, E)$  is a biclique, then any induced subgraph  $G' = (V'_1, V'_2, E')$  of  $G$  such that  $V'_1 \neq \phi$ ,  $V'_2 \neq \phi$ ,  $V'_1 \subseteq V_1$  and  $V'_2 \subseteq V_2$  is also a biclique. The bicliques induced from a biclique  $G$  provides no more information than  $G$ , so we focus on mining maximal bicliques in this paper.

**Definition 2 (Maximal biclique).** *Let  $G'$  be a biclique subgraph of graph  $G$ . If there does not exist any other biclique subgraph  $G''$  of  $G$  such that  $G'$  is a proper subgraph of  $G''$ , then  $G'$  is a maximal biclique of  $G$ .*

**Proposition 3.** *Let  $V_1$  and  $V_2$  be two sets of vertices in graph  $G = (V, E)$  and  $E' = \{\{u, v\} | u \in V_1, v \in V_2 \text{ and } \{u, v\} \in E\}$ . Graph  $G' = (V_1, V_2, E')$  is a maximal biclique subgraph of  $G$  if and only if  $\Gamma(V_1, G) = V_2$  and  $\Gamma(V_2, G) = V_1$ .*

The proof of this proposition can be found at [12].

**Corollary 1.** *Let  $V_1$  be a set of vertices in  $G = (V, E)$ .  $G' = (V_1, \Gamma(V_1, G))$  is a maximal biclique subgraph of  $G$  if and only if  $\Gamma(\Gamma(V_1, G), G) = V_1$ .*



**Fig. 1.** The search space

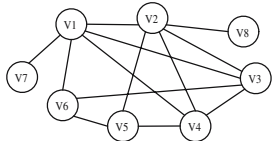
A maximal biclique  $G'$  is called a **large maximal biclique** if the size of its both vertex sets is no less than a predefined threshold  $ms$ . The task of mining large maximal biclique subgraphs is to enumerate all the large maximal biclique subgraphs with respect to  $ms$  from a given graph.

Given a graph  $G = (V, E)$ , any subset of  $V$  can form a biclique with another subset of  $V$ . Therefore, the search space of the large maximal biclique subgraph mining problem is the power set of  $V$ . Figure 1 shows the search space of a graph with five vertices  $\{1, 2, 3, 4, 5\}$ . We are interested in only those large maximal bicliques. A vertex set is not of interest if itself is too small or its adjacency list is too small. Therefore, in Figure 1, only those vertex sets between the two borders (indicated by dotted lines) are of interest. The vertex sets above the two borders are uninteresting because themselves are too small, and the vertex sets below the two borders are uninteresting because their adjacency lists are too small. We use the following proposition and Proposition 1 to prune uninteresting bicliques.

**Proposition 4.** *If a vertex set  $|V_1| < ms$ , then  $\forall V' \subset V_1$ , we have  $|V'| < ms$ .*

Proposition 4 implies that if a vertex set is smaller than the minimum size threshold, then there is no need to consider the subsets of the vertex set. This proposition can be used to prune the uninteresting vertex sets above the two borders. Proposition 1 implies that if the adjacency list of a vertex set is smaller than the minimum size threshold, then there is no need to consider the supersets of the vertex set because their adjacency lists are also smaller than the minimum size threshold. This proposition can be used to prune those uninteresting vertex sets below the two borders.

The maximal biclique mining problem is related to the frequent closed itemset mining problem [24,12]. A graph  $G$  can be mapped to a transaction database [1], denoted as  $D(G)$ , by treating *each vertex* as an item and the *adjacency list* of each vertex as a transaction. Table 1 shows a mapping example. In the frequent itemset mining problem, the concept of frequent closed itemsets is proposed to remove redundant itemsets [16]. An itemset  $X$  is *closed* if there does not exist another itemset  $Y$  such that  $Y$  is a superset of  $X$  and  $T(Y) = T(X)$ , where  $T(X)$  denotes the set of transactions containing  $X$ , that is,  $T(X) = \{t | t \in D(G), X \subseteq t\}$ . This definition is equivalent to Definition 2 because  $T(X) = \{t | t \in D(G),$

**Table 1.** A graph  $G$  is mapped into a transaction database  $D(G)$ 

 $\Rightarrow$ 

TID	Transactions
$v_1$	$v_2, v_3, v_4, v_6, v_7$
$v_2$	$v_1, v_3, v_4, v_5, v_8$
$v_3$	$v_1, v_2, v_4, v_6$
$v_4$	$v_1, v_2, v_3, v_5$
$v_5$	$v_2, v_4, v_6$
$v_6$	$v_1, v_3, v_5$
$v_7$	$v_1$
$v_8$	$v_2$

$X \subseteq t\}$  defined on the transaction database is equivalent to  $\Gamma(X) = \{u|u \in G, X \subseteq \Gamma(u)\}$  defined on the graph.

Li et al. [12] has proved that there is a correspondence between maximal bicliques and closed itemsets, that is, a maximal biclique in  $G$  corresponds to a pair of closed itemsets in  $D(G)$ . Based on this observation, mining large maximal bicliques with respect to  $ms$  from a graph  $G$  is equivalent to mining frequent closed itemsets with respect to  $ms$  from  $D(G)$ , and the size of the frequent itemsets should be no less than  $ms$ . However, using existing frequent closed itemset mining algorithms [23,22,13,20] to mine large maximal bicliques has several drawbacks. First, the large maximal biclique mining problem has size constraints on both vertex sets, but existing closed itemset mining algorithms only use the size constraint on transaction sets to prune the search space, which not only generate many uninteresting small maximal bicliques but also waste mining cost. Secondly, a maximal biclique corresponds to a pair of closed itemsets, so each maximal biclique is generated twice using frequent closed itemset mining algorithms. Finally, frequent itemset mining algorithms produce only itemsets, so a post-processing step is necessary to obtain the corresponding transaction set for each frequent closed itemset. The post-processing step can be costly when both the number of closed itemsets and the transaction database are very large. Algorithms that adopt the vertical mining approach can be modified to produce both itemsets and transaction sets during the mining process, but they still have the first two drawbacks. In the next section, we present an algorithm which mine maximal bicliques directly.

### 3 New Algorithm for Mining Large Maximal Bicliques

Given a graph  $G = (V, E)$ , the search space of the large maximal biclique mining problem is the power set of  $V$ . The search space can be represented by a set enumeration tree as shown in Figure 1. The root of the tree represents the empty set. Each node at level  $k$  represents a vertex set containing  $k$  vertices. The subtree rooted at vertex set  $X$  is called the sub search space tree of  $X$ . In a search space tree, the vertices are sorted into some order. For every vertex set  $X$  in the tree, only vertices after the last vertex of  $X$  can appear in the sub search space tree of  $X$ . This set of vertices are called *tail vertices* of  $X$ , denoted

as  $tail(X)$ . For example, in the search space tree shown in Figure 11, vertices are sorted into lexicographic order, so vertex 4 is in  $tail(\{1, 3\})$ , but vertex 2 is not a tail vertex of  $\{1, 3\}$  because vertex 2 is before vertex 3.

In a search space tree, the search space of every internal vertex set is partitioned into several disjoint sub search spaces by the child nodes of the vertex set. We explore the search space tree in depth-first order to recursively partition the whole search space into small sub search spaces. At each node, we generate the adjacency list of the corresponding vertex set  $X$ . Based on Proposition 1, if the size of the adjacency list is less than the predefined size threshold  $ms$ , the search on that branch should be terminated to avoid mining those vertex sets whose adjacency lists are smaller than  $ms$ . If the adjacency list of a vertex set is no less than  $ms$ , we call the vertex set as *frequent*. For each frequent vertex set  $X$ , we identify those vertices  $v$  from  $tail(X)$  such that  $|\Gamma(X \cup \{v\})| \geq ms$ , and the child nodes of  $X$  representing these vertices should be explored further. The mining is performed on these child nodes recursively.

The vertex sets that themselves are too small are pruned based on Proposition 4. The vertex sets appearing in the sub search space of a vertex set  $X$  are subsets of  $X \cup tail(X)$ . Based on Proposition 4, if  $|X| + |tail(X)|$  is less than  $ms$ , then there is no need to search in the subtree rooted at  $X$  because all the vertex sets in that subtree contain less than  $ms$  vertices. Similarly, if there are less than  $ms - |V|$  vertices  $v \in Tail(X)$  such that  $|\Gamma(X \cup \{v\})| \geq ms$ , then there is no need to search in the subtree rooted at  $X$  either.

Algorithm 1 shows the pseudo-codes of the mining algorithm. When the algorithm is first called on a graph  $G = (V, E)$ ,  $X$  is set to the empty set, and  $\Gamma(X)$  and  $tail(X)$  are set to  $V$ . For a vertex set  $X$ , we first remove those vertices  $v$  from  $tail(X)$  such that the adjacency list of  $X \cup \{v\}$  is less than  $ms$  (line 1-3). Then we check whether  $|X| + |tail(X)|$  is less than  $ms$ . If it is true, then the search should be terminated based on Proposition 4 (line 4-5). If  $|X| + |tail(X)|$  is no less than  $ms$ , then the algorithm is recursively called for each  $v \in tail(X)$  (line 7-14). Before we search in the sub search space tree of  $X \cup v$ , we check whether  $|X \cup \{v\}| + |tail(X \cup \{v\})|$  is no less than  $ms$ . Only if  $|X \cup \{v\}| + |tail(X \cup \{v\})|$  is no less than  $ms$ , the search in the sub search space tree of  $X \cup v$  should continue (line 9).

We sort the vertices in  $tail(X)$  into ascending order of  $|\Gamma(X \cup \{v\})|$  (line 6), that is, for any two vertices  $u, v \in tail(X)$ , if  $|\Gamma(X \cup \{u\})| > |\Gamma(X \cup \{v\})|$ , then  $u$  is after  $v$  in the order. The ascending ordering method has been adopted in many frequent itemset mining algorithms and has been proved to be very effective for pruning the search space. The rationale behind this ordering method is to let the vertex set with a smaller adjacency list have a larger number of tail vertices and the vertex set with a larger adjacency list have a smaller number of tail vertices so that the vertex sets in the sub search space tree of the vertex set with a smaller adjacency list are likely to be pruned because of their small adjacency lists, and the vertex sets in the sub search space tree of the vertex set with a larger adjacency list are likely to be pruned because of their small tail vertex sets.

**Algorithm 1.** MineLMBC Algorithm**Input:**

$X$  is a vertex set  
 $\Gamma(X)$  is the adjacency list of  $X$   
 $tail(X)$  is the tail vertices of  $X$   
 $ms$  is the minimum size threshold;

**Description:**

```

1: for all vertex  $v \in tail(X)$  do
2:   if  $|\Gamma(X \cup \{v\})| < ms$  then
3:      $tail(X) = tail(X) - \{v\}$ 
4: if  $|X| + |tail(X)| < ms$  then
5:   return ;
6: Sort vertices in  $tail(X)$  into ascending order of  $|\Gamma(X \cup \{v\})|$ ;
7: for all vertex  $v \in tail(X)$  do
8:    $tail(X) = tail(X) - \{v\}$ ;
9:   if  $|X \cup \{v\}| + |tail(X)| \geq ms$  then
10:     $Y = \Gamma(\Gamma(X \cup \{v\}))$ ;
11:    if  $((Y - (X \cup \{v\})) \subseteq tail(X))$  then
12:      if  $|Y| \geq ms$  then
13:        Output  $(Y, \Gamma(X \cup \{v\}))$  as a large maximal biclique;
14:      MineLMBC( $Y, \Gamma(X \cup \{v\}), tail(X) - Y, ms$ );

```

One optimization can be made to Algorithm 1 is to prune the adjacency list of a vertex set based on Proposition 4. Let  $v$  be a vertex in  $\Gamma(X)$ . If  $v$  is adjacent to less than  $ms - |X|$  vertices in  $tail(X)$ , then  $v$  cannot be adjacent to any superset  $Y$  of  $X$  such that  $|Y| \geq ms$ . Hence vertex  $v$  can be removed from  $\Gamma(X)$ . By pruning the adjacency list, those vertex sets with a small adjacency list can be identified and pruned earlier.

**Pruning non-maximal bicliques.** Non-maximal bicliques are identified and pruned during the mining process. Let  $X$  be a vertex set in the search space tree. Based on Corollary 1, biclique  $G' = (X, \Gamma(X))$  is maximal if and only if  $\Gamma(\Gamma(X)) = X$  is true. If  $G'$  is not a maximal biclique, that is,  $\Gamma(\Gamma(X)) \neq X$ , then  $\Gamma(\Gamma(X))$  must be a proper superset of  $X$  based on Proposition 2. We can prune the sub search space of  $X$  based on the following proposition.

**Proposition 5.** *Let  $X$  be a vertex set. For any maximal biclique  $G' = (V_1, V_2)$  such that  $X \subseteq V_1$ , we have  $\Gamma(\Gamma(X)) \subseteq V_1$ .*

*Proof.* Biclique  $G'$  is maximal, so we have  $V_2 = \Gamma(V_1)$ . Vertex set  $X$  is a subset of  $V_1$ , so we have  $V_2 = \Gamma(V_1) \subseteq \Gamma(X)$  based on Proposition 1, and hence based on Proposition 1 again, we have  $\Gamma(\Gamma(X)) \subseteq \Gamma(V_2) = V_1$ .

The above proposition indicates that if a maximal biclique has a vertex set containing  $X$ , then the vertex set must also contain  $\Gamma(\Gamma(X))$ . If biclique  $G' = (X, \Gamma(X))$  is not maximal, there are two cases. One case is that  $\Gamma(\Gamma(X)) - X$  is a subset of  $tail(X)$ . For this case, the maximal bicliques that have a vertex set containing  $X$  are in the sub search space of  $X$ . Since any maximal biclique that have a vertex set containing  $X$  must also contain  $\Gamma(\Gamma(X))$ , we use  $\Gamma(\Gamma(X))$  to replace  $X$  and remove vertices in  $\Gamma(\Gamma(X)) - X$  from  $tail(X)$  to prune those non-maximal bicliques that have a vertex set containing  $X$  but not containing  $\Gamma(\Gamma(X))$ . The other case is that there exists a vertex  $v \in (\Gamma(\Gamma(X)) - X)$  such that  $v$  is not in  $tail(X)$ . For this case, none of the bicliques discovered from the sub search space tree of  $X$  can be maximal because these bicliques have a

vertex set containing  $X$  but this vertex set does not contain  $v$ . To avoid mining non-maximal bicliques, we check whether  $\Gamma(\Gamma(X)) = X$  is true before searching in the sub search space tree of  $X$ . If it is not true and there exists a vertex  $v \in \Gamma(\Gamma(X))$  such that  $v$  is not in  $\text{tail}(X)$ , then we skip the sub search space tree of  $X$  (line 11).

**Pruning duplicate bicliques.** A biclique has two disjoint vertex sets. Our search strategy is based on vertex set searching. Therefore, every maximal biclique is generated twice in Algorithm 1. It is desirable to avoid generating duplicate bicliques to save mining cost. We prune duplicate bicliques based on the following observation, which is inspired by the pruning technique for mining maximal bicliques [5,19]. Given a vertex  $v$  in a graph  $G = (V, E)$ , the adjacency list of any subset of  $\Gamma(v)$  must contain  $v$  because  $v$  is adjacent to all the vertices in  $\Gamma(v)$ . If we have generated all the vertex sets containing  $v$  and their adjacency lists, then there is no need to generate any subset of  $\Gamma(v)$ . To maximize the number of vertex sets being pruned, we pick the vertex with the largest adjacency list and prune all the subsets of its adjacency list.

We use the graph shown in Table 1 to illustrate the pruning of duplicate bicliques. We set  $ms$  to 2. There are 6 vertices whose adjacency list is no less than 2. We sort the 6 vertices in ascending order of their adjacency list size, and we get  $\{v_6, v_5, v_4, v_3, v_2, v_1\}$ . Vertex  $v_1$  has the largest adjacency list. We adjust the ordering by putting those vertices adjacent to  $v_1$  to the end of the ordering, and we get  $\{v_5, v_1, v_6, v_4, v_3, v_2\}$ . All the vertex sets discovered from the sub search space tree of  $v_6, v_4, v_3$  and  $v_2$  must be subsets of  $\Gamma(v_1) = \{v_2, v_3, v_4, v_6, v_7\}$ , thus their adjacency lists must contain  $v_1$ . All the vertex sets containing  $v_1$  have already been discovered from the sub search space tree of  $v_5$  and  $v_1$ . Therefore, there is no need to search in the sub search space tree of  $v_6, v_4, v_3$  and  $v_2$ .

Using the above method, many duplicate bicliques can be pruned especially when there is a vertex in the graph with a very high degree. However, we cannot guarantee that all of the duplicate bicliques can be pruned using the above method. The remaining duplicate bicliques can be identified by comparing a vertex set with its adjacency list. The adjacency list of a vertex set is also a vertex set in the search space tree. If the adjacency list of a vertex set is before the vertex set in the search space tree in depth-first order, it means that the biclique has been generated from the adjacency list before and the biclique generated from the vertex set itself is a duplicate.

When mining maximal bicliques from bipartite graphs, duplicate maximal bicliques can be completely avoided. Let  $G = (V_1, V_2, E)$  be a bipartite graph. Since there is no edge between two vertices in  $V_1$  or two vertices in  $V_2$ , the two vertex sets of a biclique discovered from  $G$  cannot both be subsets of  $V_1$  or subsets of  $V_2$ . Therefore, it is sufficient to use only one vertex set of bipartite graph  $G$  to form the search space. We pick the smaller vertex set to form the search space. The adjacency lists of the vertex sets generated during the mining process must be from the other vertex set of  $G$ , and they are never searched. Hence duplicate bicliques can be avoided.

The correctness of Algorithm 1 is guaranteed by Propositions 1, 3, 4 and 5.



**Theorem 1.** *Given a graph  $G$  and a minimum size threshold  $ms$ , Algorithm 1 generates all the large maximal biclique subgraphs with respect to  $ms$  from  $G$ , and only the large maximal biclique subgraphs of  $G$  are generated.*

For a complexity analysis on the time and space of Algorithm 1, we use the following notations:  $n$  is the total number of vertices,  $d$  the maximal degree of the vertices,  $m$  the number of edges and  $N$  the number of maximal bicliques.

In algorithm 1, before we search in the sub search space tree of a vertex set  $X$ , we first check whether  $\Gamma(\Gamma(X)) \subseteq (X \cup \text{tail}(X))$  is true. If it is not true, then there is no need to explore the sub search space tree of  $X$  because no biclique in the sub search space tree of  $X$  is maximal. Hence Algorithm 1 is called only for each maximal biclique. The cost for generating  $\Gamma(\Gamma(X))$  is bounded by  $d^2$ . When exploring the sub search space tree of a vertex set  $X$ , we first find all those vertices  $v \in \text{tail}(X)$  such that  $|\Gamma(X \cup \{v\})| \geq ms$  (line 1-3). The cost of this step is bounded by  $|\text{tail}(X)| \cdot |\Gamma(X)| \leq nd$ . The cost for sorting the vertices in  $\text{tail}(V)$  is bounded by  $|\text{tail}(X)| \cdot \log(|\text{tail}(X)|) \leq n \cdot \log n$ . Therefore, the worst-case time complexity of Algorithm 1 is  $O(nd \cdot N)$ .

During the mining process, we keep the whole graph in the memory. The space overhead for storing the graph is  $O(m)$ . When exploring the search space in depth-first order, we need to keep the adjacency lists of the vertex sets on the path this is currently being visited. The maximal depth of the path is bounded by  $d$  and the maximal size of the adjacency list of a vertex set is also bounded by  $d$ . Hence the space complexity of Algorithm 1 is  $O(m + d^2)$ .

## 4 A Performance Study

We conducted a set of experiments to demonstrate the efficiency and flexibility of our algorithm. Our experiments were conducted on a PC with an Intel Pentium IV 3.6GHz processor and 2GB of main memory. The operating system is Fedora Core 4. Our algorithm was implemented using C++ and compiled using g++.

We compared our algorithm with two other algorithms. The first algorithm is MICA [2], which uses consensus algorithms to generate maximal bicliques. The MICA algorithm is available at <http://genome.cs.iastate.edu/supertree/download/biclique/README.html>. The second algorithm is the current fastest closed pattern mining algorithm—LCM, which shows the best performance in a comparative study of frequent itemset mining implementations [3]. We used the latest version of LCM—LCM3 [21] in our experiments, which is kindly provided by Takeaki Uno. Table 2 shows some information of three graphs used in our experiments. All of them are obtained from the Second DIMACS Challenge benchmarks<sup>1</sup>. Here, the edge density of a graph is the number of edges of the graph divided by the total possible number of edges of the graph.

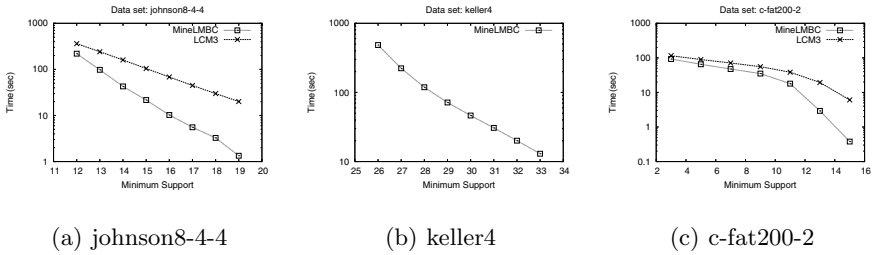
We observed that most graphs in the Second DIMACS Challenge benchmarks are very dense and all the algorithms takes a long time to finish. Here we report results on three graphs shown in Table 2, on which at least one of the algorithms

<sup>1</sup> <ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/cliique/>

**Table 2.** Three dense graphs

Graph	#vertices $n$	#Edges	edge density
johnson8-4-4	70	1855	0.768
keller4	171	9435	0.649
c-fat200-2	200	3235	0.163

can terminate within one hour. For sparse graphs in the benchmarks, such as johnson8-2-4, c-fat200-1, c-fat500-1 and p.hat300-1, all the algorithms can terminate within several seconds, so we do not show the results on these sparse graphs due to the limit of space.



**Fig. 2.** Running time (The running time of MICA exceeds one hour on all the datasets, so it is not shown in the figures.)

Figure 2 shows the running time (y-axis) of the three algorithms with respect to the minimum size threshold (x-axis). The MICA algorithm cannot complete the mining task in any of the three datasets within one hour. LCM is unable to complete the task for keller4 because keller4 has a high edge density of 0.649 and a large number vertices of 171. Only our algorithm (denoted mineLMBC in the figure) is able to enumerate the large maximal bicliques from all the three graphs within one hour. The similar performance of MineLMBC and LCM3 in graph c-fat200-2 may be attributed to the fact that c-fat200-2 has a much lower edge density than the other two graphs.

## 5 Related Work

Enumerating all maximal bicliques from graphs is a NP-complete problem [2]. Some related problems have been studied. The maximal vertex biclique problem is to decide whether or not a bipartite graph contains a biclique such that  $|V_1| + |V_2| \geq k$ , and it can be solved in polynomial time [10]. If the constraint is that  $|V_1| = |V_2| = k$  (this is called the balanced biclique problem) or  $|V_1| \cdot |V_2| = k$  (this is called the maximal edge biclique problem), then the problem is NP-complete [10,17]. Geerts et al. [9] developed an approximate algorithm to find tilings—a collection of tiles from transaction databases, where a tile is a region in the database consisting solely of ones. Besson et al. [4] proposed an algorithm D-Miner to compute constrained concepts, i.e., closed sets and

associated transaction sets. Mishra et al. [15] propose a new formulation of the conceptual clustering problem where the goal is to explicitly output a collection of simple and meaningful conjunctions of attributes that define the clusters. Connections between this conceptual clustering problem and the maximum edge biclique problem are made. Randomized algorithms are given that discover a collection of approximate conjunctive cluster descriptions in sublinear time.

Several algorithms have been proposed to mine maximal bicliques. Makino et al. [14] propose three algorithms to mine bicliques from bipartite graphs. The first algorithm runs with  $O(M(n) \cdot N)$  time complexity and  $O(n^2)$  space, the second one runs with  $O(d^3 \cdot N)$  time complexity and  $O(n + m)$  space, and the last one runs with  $O(d^2 \cdot N)$  time complexity and  $O(n + m + d \cdot N)$  space, where  $n$  is the number of vertices,  $M(n)$  is the time needed to multiply two  $n \times n$  matrices,  $m$  is the number of edges,  $d$  is the maximal degree of vertices and  $N$  is the number of maximal bicliques. Eppstein [8] proves that all maximal bipartite cliques can be enumerated in time  $O(a^3 \cdot 2^{2a} \cdot (n+m))$ , where  $a$  is the minimum number of forests into which the edges of the graph can be partitioned and it can easily be around 10 to 20 in practice. Alexe et al. [2] use consensus algorithms to mine maximal bicliques. Their algorithms need to keep all the maximal bicliques in memory, so the space complexity of their algorithm is  $O(N)$ , and the time complexity of their algorithm is  $O(n^3 \cdot N)$ . Our algorithm has better complexities than the above algorithms. Furthermore, the algorithms proposed by Makino et al. are limited to bipartite graphs. Our algorithm can be applied to any undirected graphs.

## 6 Conclusion

In this paper, we have presented an efficient algorithm for mining large maximal biclique subgraphs from undirected graphs. The proposed algorithm explores the search space in depth-first order. It effectively utilizes size constraints on both vertex sets to prune the search space. Our performance study shows that the proposed algorithm outperforms previous algorithms significantly.

## References

1. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc of the 1993 ACM SIGMOD Conference*, pages 207–216, 1993.
2. 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*, 145(1):11–21, 2004.
3. R. J. Bayardo, B. Goethals, and M. J. Zaki, editors. *Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
4. J. Besson, C. Robardet, and J.-F. Boulicaut. Constraint-based mining of formal concepts in transactional data. In *Proc. of the 8th PAKDD Conference*, pages 615–624, 2004.

5. C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
6. D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. Topological structure analysis of the protein protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450, 2003.
7. F. Chung. On the coverings of graphs. *Discrete Applied Mathematics*, 30(2):89–93, 1980.
8. D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4), 1994.
9. B. G. Floris Geerts and T. Mielikäinen. Tiling databases. In *Proc. of the 7th International Conference on Discovery Science*, pages 278–289, 2004.
10. M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.
11. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *Proceeding of the 8th international conference on World Wide Web*, pages 1481–1493, 1999.
12. J. Li, H. Li, D. Soh, and L. Wong. A correspondence between maximal complete bipartite subgraphs and closed patterns. In *Proc. of the 9th PKDD Conference*, pages 146–156, 2005.
13. G. Liu, H. Lu, W. Lou, and J. X. Yu. On computing, storing and querying frequent patterns. In *Proc. of the 9th ACM SIGKDD Conference*, pages 607–612, 2003.
14. K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Proc. of the 9th Scandinavian Workshop on Algorithm Theory*, pages 260–272, 2004.
15. N. Mishra, D. Ron, and R. Swaminathan. A new conceptual clustering framework. *Machine Learning*, 56(1-3), 2004.
16. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th ICDT Conference*, pages 398–416, 1999.
17. R. Peeters. The maximum edge biclique problem is np-complete. Research Memorandum 789, Tilburg University, Faculty of Economics and Business Administration, 2000.
18. M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley. Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular Biology and Evolution*, 20(7):1036–1042, 2003.
19. E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques. In *International Computing and Combinatorics Conference (COCOON 2004)*, pages 161–170, 2004.
20. T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proc. of the ICDM 2004 Workshop on Frequent Itemset Mining Implementations*, 2004.
21. T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proc. of the ACM SIGKDD Open Source Data Mining Workshop on Frequent Pattern Mining Implementations*, 2005.
22. J. Wang, J. Pei, and J. Han. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. of the 9th ACM SIGKDD Conference*, pages 236–245, 2003.
23. M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proc. of SIAM International Conference on Data Mining*, pages 398–416, 2002.
24. M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *Proc. of the 3rd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998.

知网查重限时 7折 最高可优惠 120元

本科定稿，硕博定稿，查重结果与学校一致

立即检测

论文查重: <http://www.paperyy.com>

文献下载: <http://www.ixueshu.com>

论文自动降重: [http://www.paperyy.com/reduce\\_repetition](http://www.paperyy.com/reduce_repetition)

模版下载: <http://ppt.ixueshu.com>

---