Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A One-to-one Correspondence and Mining Algorithms

Jinyan Li, *Member, IEEE,* Guimei Liu, Haiquan Li, and Limsoon Wong

***Abstract*— Maximal biclique (also known as complete bipartite) subgraphs can model many applications in web mining, business, and bioinformatics. Enumerating maximal biclique subgraphs from a graph is a computationally challenging problem, as the size of the output can become exponentially large with respect to the vertex number when the graph grows. In this paper, we efficiently enumerate them through the use of closed patterns of the adjacency matrix of the graph. For an undirected graph G without self-loops, we prove that: (i) the number of closed patterns in the adjacency matrix of G is even; (ii) the number of the closed patterns is precisely double the number of maximal biclique subgraphs of G; and (iii) for every maximal biclique subgraph, there always exists a unique pair of closed patterns that matches the two vertex sets of the subgraph. Therefore, the problem of enumerating maximal bicliques can be solved by using efficient algorithms for mining closed patterns, which are algorithms extensively studied in the data mining field. However, this direct use of existing algorithms causes a duplicated enumeration. To achieve high efficiency, we propose an $O(mn)$ time delay algorithm for a non-duplicated enumeration, in particular for enumerating those maximal bicliques with a large size, where m and n are the number of edges and vertices of the graph respectively. We evaluate the high efficiency of our algorithm by comparing it to state-of-the-art algorithms on three categories of graphs: randomly generated graphs, benchmarks, and a real-life protein interaction network. In this paper, we also prove that if self-loops are allowed in a graph, then the number of closed patterns in the adjacency matrix is not necessarily even; but the maximal bicliques are exactly the same as those of the graph after removing all the self-loops.**

***Index Terms*— Maximal biclique subgraphs, closed patterns, mining methods and algorithms, bioinformatics (genome or protein) database.**

I. Introduction

Graph mining has recently become an important and active research topic in the data mining field. Indeed, there are plenty of prior work on mining frequent subgraphs [1], [2], [3], [4], [5], [6], on mining patterns from graph databases [7], [8], [9], [10], [11], on mining dense subgraphs or important quasi-cliques crossing graphs [12], [13], on mining closed graph patterns [14], [15], etc. (See [16] for a survey on earlier graph mining research work.) In this paper, we investigate efficient algorithms for mining *maximal biclique subgraphs* from a large graph. Given a graph G, a maximal biclique subgraph consists of two disjoint subsets of vertices of G that exhibit a full connectivity between the two vertex groups. This subgraph concept emphasizes the interaction between the two groups of vertices. In contrast, the notion of cliques and quasi-cliques reflects just the density of edges within one group of vertices or the compactness of this group, and the notion of frequent subgraphs captures only the occurrence of subgraphs in a given graph or a graph database.

Many real-life applications can be modeled by maximal bicliques. Here we give two examples. The first one is from social networks. Web communities have been modeled by bipartite cores [17], [18], [19], [20], and they can be discovered through identifying maximal bicliques from web networks. This idea can be easily extended to mobile communication networks to find out interacting customer communities which perhaps are commercially useful. The second example is about protein interactions in the biological field. The thousands of proteins in a biological cell can be modeled by a graph with a vertex representing a protein and an edge representing an interaction between a pair of proteins. The maximal biclique subgraphs from such protein interaction graphs are critical to questions such as which two protein groups have a full interaction. This is an important problem in bioinformatics and biology [21], [22], [23], [24] for a variety of purposes, such as for function inference of unknown proteins, for discovery of binding motif pairs, and for study of topological sub-structures of protein interaction networks. The same mining problem has been also observed in the reconstruction of the supertree of Life, the central topic in the phylogenetics studies [25], [26], [27], where co-clusterings (bicliques) between groups of species and genes are demanded.

The enumeration of maximal bicliques from a graph has been long studied [28], [29], [30]. Eppstein [29] has proposed an algorithm with the time complexity of $O(a^3 \cdot 2^{2 \cdot a} \cdot n)$, where n is the number of vertices of G, and a the arboricity of the graph—the minimum number of forests into which the edges of G can be partitioned. Since the number a can be easily large, this linear complexity algorithm is actually not efficient for large graphs. Makino's three algorithms [30] all deal with the mining of biclique subgraphs from *bipartite graphs*, which are a special type of graph. [1] Dias et al. [31] had an excellent work on listing *induced* maximal bicliques. As an induced maximal biclique strictly forbids any intra-edges within the two vertex sets,

Manuscript received August 31, 2006; revised March 15, 2007 and August 3, 2007.

J. Li is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798. Email: jyli@ntu.edu.sg. Correspondence author.

G. Liu is with the School of Computing, National University of Singapore. Email: liugm@comp.nus.edu.sg.

H. Li is with the Institute for Infocomm Research, Singapore. Email: Haiquan@alumni.nus.edu.sg.

L. Wong is with the School of Computing, National University of Singapore. Email: wongls@comp.nus.edu.sg.

[1] While it is possible to modify these algorithms to handle general graphs, we were unable to obtain the source codes from Makino et al. for this modification and comparison purposes.

non-induced maximal bicliques (our work) are more suitable for real-life applications such as those mentioned above. Unlike our approach transforming a graph into a transactional database [32] for subgraph enumeration, Zaki and Ogihara had an innovative work [33] on how a transactional database can be converted into a bipartite graph for itemset discovery. Only the work by Alex et al. [28] has studied the same problem as ours. Their consensus algorithm is called MICA, with the time complexity of $O(n^3 \cdot N)$ and the space complexity of $O(N)$, where N is the number of maximal bicliques.

In this paper, we introduce a new algorithm with the time complexity of $O(mn{\cdot}N)$ and the space complexity of only $O(mn)$ to enumerate maximal bicliques from large graphs, where m is the number of edges of the graph. Unlike the MICA algorithm, our algorithm does not have to store all the maximal bicliques in memory. Our method is based on the novel observation that enumerating all maximal biclique subgraphs from a graph is equivalent to the problem of mining *closed patterns* [34] from the adjacency matrix of this graph. This observation is derived from the following two propositions. For an undirected graph G without self-loops:

- The number of the closed patterns in the adjacency matrix of G is precisely double the number of maximal biclique subgraphs of G;
- For every maximal biclique subgraph, there always exists a unique pair of closed patterns that matches the two vertex sets of the subgraph.

Many prior algorithms and implementations have been developed for efficient mining of closed patterns, e.g., the pioneering A-close algorithm [34], the later CHARM [35] and CLOSET+ [36], and the more recent FPclose [37], [38] and LCM [39]. Each of these can be directly applied for the enumeration. However, one drawback with such a direct use is that the maximal bicliques will be all enumerated twice. Therefore, we modify the LCM [39] algorithm to develop our new algorithm, called LCM-MBC, tailoring for a non-redundant enumeration of maximal bicliques from a large and dense graph. As not all maximal bicliques are equally interesting, we adapt LCM-MBC to discover only (p, q)-large maximal biclique subgraphs without enumerating small maximal bicliques. To evaluate the high efficience of LCM-BMC, we not only test and compare our algorithm on randomly generated and benchmark graph datasets, but also on a real-life protein interaction network.

The rest of the paper is organized as follows: Sections II and III provide basic definitions and properties for graphs and closed patterns. Section IV presents a proof for the one-to-one correspondence between closed pattern pairs and maximal biclique subgraphs of a graph. Section 5 presents our main algorithm, LCM-MBC, for mining maximal bicliques whose vertex size is larger than a threshold. Section VI shows our experimental results at two aspects: (i) comparing the efficiency of our LCM-MBC algorithm to that of LCM [39] and MICA [28]; (ii) reporting maximal biclique subgraphs discovered from a protein interaction graph, and using them to explain and re-confirm the so-called "many-few" property of protein interaction networks. Section 7 discusses how to determine and enumerate maximal biclique subgraphs when self-loops are allowed in a graph. Section 8 discusses relationships with formal concept analysis [40], and biclustering techniques [41]. Section 9 concludes this paper with a summary and a future work.

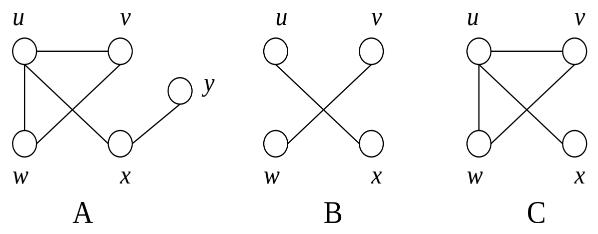


Fig. 1. Three graphs and their relationships.

II. Maximal Biclique Subgraphs: The Background

A graph $G = \langle V^G, E^G \rangle$ comprises a set of vertices V^G and a set of edges $E^G \subseteq V^G \times V^G$. Through this paper, we assume G is an undirected graph without self-loops (unless specified otherwise); namely, no edge $(u, u) \in E^G$, and every $(u, v) \in E^G$ is an unordered pair. The superscripts in V^G and E^G can be omitted when the context is clear.

A graph H is a subgraph of a graph G iff $V^H \subseteq V^G$ and $E^H \subseteq E^G$. Let S be a non-empty set of vertices of a graph G. The subgraph *induced* by S is the maximal subgraph of G with vertex set S, denoted by $\langle S \rangle^G$. So, $\langle S \rangle^G$ contains precisely those edges of G joining all possible two vertices in S. Subgraph $\langle S \rangle^G$ is called a *vertex-induced subgraph* of G, or simply, an induced subgraph of G. A graph G is *bipartite* if V^G can be partitioned into two disjoint non-empty subsets V_1 and V_2 such that $E^G \subseteq V_1 \times V_2$. In this case, the graph G is usually denoted by $G = \langle V_1 \cup V_2, E^G \rangle$. Note that there is no edge in G that joins any two vertices within V_1 or V_2. G is a *complete bipartite*, or called a *biclique*, if $E^G = V_1 \times V_2$.

Suppose H is a bipartite subgraph of a graph G, then the subgraph induced by V^H may not be a bipartite subgraph of G. So, the number of induced bipartite subgraphs cannot be larger than—and it is often less than—that of bipartite subgraphs in a graph. In this paper, we focus on the normal bipartite subgraphs, rather than vertex-induced bipartite subgraphs [31].

Example 1: Figure 1 shows three graphs A, B, and C, where

$$\begin{aligned} A &= \langle \{u, v, w, x, y\}, \{(u, v), (u, x), (u, w), (w, v), (x, y)\} \rangle \\ B &= \langle \{u, v\} \cup \{w, x\}, \{(u, x), (w, v)\} \rangle \\ C &= \langle \{u, v, w, x\}, \{(u, v), (u, w), (u, x), (w, v)\} \rangle \end{aligned}$$

Then B is a bipartite subgraph of A, but it is not a vertex-induced subgraph from the vertex set $\{u, v, w, x\}$. On the other hand, C is a vertex-induced subgraph from the vertex set $\{u, v, w, x\}$, but C is not a bipartite subgraph of A.

Two vertices u, v of a graph G are said to be adjacent if $(u, v) \in E^G$. The *neighborhood* $\beta^G(v)$ of a vertex v of a graph G is the set of all vertices in G that are adjacent to v, namely, $\beta^G(v) = \{u \mid (u, v) \in E^G\}$. The neighborhood $\beta^G(X)$ for a *non-empty* subset X of vertices of a graph G is the set of common neighborhoods of the vertices in X. That is, $\beta^G(X) = \bigcap_{x \in X} \beta^G(x)$.

For every non-empty subset X of vertices in a graph G such that $\beta^G(X)$ is also non-empty, then it is the case that $H = \langle X \cup \beta^G(X), X \times \beta^G(X) \rangle$ is a biclique subgraph of G. It is possible that a vertex $v \notin X$ of G can be adjacent to every vertex of $\beta^G(X)$. In this case, the subset X can be expanded by adding the vertex v, while maintaining the same neighborhood. The intuition behind this expansion can be used to define the concept of *maximal biclique subgraphs*.

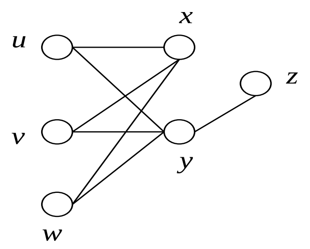


Fig. 2. A graph and a maximal biclique subgraph.

Definition 1: A biclique subgraph $H = \langle V_1 \cup V_2, E\rangle$ is a maximal biclique subgraph of G if $\beta^G(V_1) = V_2$ and $\beta^G(V_2) = V_1$.

This maximality is in the sense that there is no other biclique subgraph $H' = \langle V_1' \cup V_2', E'\rangle$ of G with $V_1 \subset V_1'$ and $V_2 \subset V_2'$, or with $V_1 \subset V_1'$ and $V_2 = V_2'$, or $V_1 = V_1'$ and $V_2 \subset V_2'$, that can satisfy $\beta^G(V_1') = V_2'$ and $\beta^G(V_2') = V_1'$. The following proposition proves this maximality.

Proposition 1: Let $H = \langle V_1 \cup V_2, E\rangle$ and $H' = \langle V_1' \cup V_2', E'\rangle$ be two maximal biclique subgraphs of G such that $V_1 \subseteq V_1'$ and $V_2 \subseteq V_2'$. Then $H = H'$.

Proof: Since $V_1 \subseteq V_1'$ and $V_2 \subseteq V_2'$, we have $\beta^G(V_1') \subseteq \beta^G(V_1)$ and $\beta^G(V_2') \subseteq \beta^G(V_2)$. Using the definition of maximal biclique subgraphs, we derive $V_2' = \beta^G(V_1') \subseteq \beta^G(V_1) = V_2$ and $V_1' = \beta^G(V_2') \subseteq \beta^G(V_2) = V_1$. Then $E = V_1 \times V_2 = V_1' \times V_2' = E'$. Thus $H = H'$ as desired. ■

Example 2: The graph shown in Figure 2 contains a maximal biclique subgraph $H = \langle\{u, v, w\} \cup \{x, y\}, \{(u, x), (u, y), (v, x), (v, y), (w, x), (w, y)\}\rangle$. Note that the subgraph

$$\langle\{u, v\} \cup \{x, y\}, \{(u, x), (u, y), (v, x), (v, y)\}\rangle$$

is a biclique, but it is not maximal.

III. Closed Patterns of an Adjacency Matrix

A graph can be equivalently described by its adjacency matrix. Let G be a graph with $V^G = \{v_1, v_2, \ldots, v_p\}$. The *adjacency matrix* $\mathbf{A}$ of G is a $p \times p$ matrix defined by

$$\mathbf{A}[i, j] = \begin{cases} 1 & \text{if } (v_i, v_j) \in E^G \\ 0 & \text{otherwise} \end{cases}$$

Recall that our graphs do not have self-loops and are undirected. Thus $\mathbf{A}$ is a symmetric matrix and every entry on the main diagonal is 0. So, $\{v_j \mid \mathbf{A}[k, j] = 1, 1 \leq j \leq p\} = \beta^G(v_k) = \{v_j \mid \mathbf{A}[j, k] = 1, 1 \leq j \leq p\}$.

A. Transformation from a graph to a special transactional database

The adjacency matrix of a graph can be transformed into a transactional database (*DB*) [32]. For ease of understanding, we review definitions related to transactional databases. A *DB* is a non-empty multi-set of transactions, a *transaction* is a subset of a pre-specified set I of *items*. Each transaction T in a *DB* is assigned a unique identity $id(T)$. A *pattern*, or called an *itemset*, is defined as a non-empty set[2] of items of I. Given a *DB* and a pattern P, the number of transactions in *DB* containing P is called the *support* of P, denoted $sup^{DB}(P)$. A pattern P is *frequent* if $sup^{DB}(P) \geq ms$, for a threshold $ms > 0$. In this paper, unless mentioned otherwise, we consider all patterns with a non-zero support, namely all those frequent patterns with the support threshold $ms = 1$. So, by a pattern of a *DB*, we mean that it is non-empty and it occurs in *DB* at least once.

[2]The $\emptyset$ is usually defined as a valid pattern in the data mining community. However, in this paper, to be consistent to the definition of $\beta^G(X)$, it is excluded.

Let G be a graph with $V^G = \{v_1, v_2, \ldots, v_p\}$. If every vertex v_i in V^G is defined as an item, then its neighborhood $\beta^G(v_i)$ is a transaction. Thus,

$$\{\beta^G(v_1), \beta^G(v_2), \ldots, \beta^G(v_p)\}$$

is a *DB*. Such a *DB* is specially denoted by DB_G. Observe that DB_G has the same number of items and transactions; and that item v_i is never contained in transaction $\beta^G(v_i)$. In this paper, we use "a pattern of DB_G" or "a pattern of the adjacency matrix of G" interchangeably.

Definition 2: The identity of a transaction $\beta^G(v_i)$ in DB_G is defined as v_i, namely, $id(\beta^G(v_i)) = v_i$.

Example 3: Figure 3 illustrates how a graph G is described by its adjacency matrix, and how it is transformed into a transactional database DB_G.

B. Closed patterns and occurrence sets

Definition 3: Let I be a set of items, and *DB* be a transactional database defined on I. Let P be a pattern, then $CL^{DB}(P) = g(f^{DB}(P))$ is defined as the **closure** of P, where $f^{DB}(P) = \{T \in DB \mid P \subseteq T\}$—all transactions in *DB* containing the pattern P; $g(D') = \bigcap_{T \in D'} T = \bigcap D'$—the set of items which are shared by all transactions in D' for a $D' \subseteq DB$.

A pattern P is a *closed pattern* of *DB* iff $CL^{DB}(P) = P$. It is already known that the adjacency matrix of a graph G can be transformed into a special transactional database DB_G, as shown in Figure 3. Therefore, we can mine closed patterns from every DB_G. A brute-force approach to mining closed patterns from DB_G is to directly use one of existing closed pattern mining algorithms [42], [43], [38], [34], [39], [36], [35].

We also define the *occurrence set* of a pattern P in DB_G:

Definition 4: Let G be a graph. The occurrence set of a pattern P in DB_G, denoted $occ^{DB_G}(P)$, is defined as $occ^{DB_G}(P) = \{id(T) \mid T \in DB_G, P \subseteq T\} = \{id(T) \mid T \in f^{DB_G}(P)\}$.

In other words, $id(T) \in occ^{DB_G}(P)$ iff $T \in f^{DB_G}(P)$, or, $v \in occ(P)$ iff v is adjacent to every vertex in P.

Proposition 2: Given a graph G and a pattern P of DB_G. Then $occ^{DB_G}(P) = \beta^G(P)$.

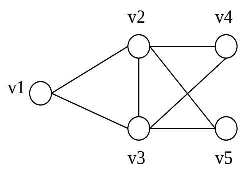
Proof: To prove the left-to-right direction of the proposition, we suppose $v \in occ(P)$. Then v is adjacent to every vertex in P. Then $v \in \beta(v')$ for each $v' \in P$. Thus $v \in \bigcap_{v' \in P} \beta(v') = \beta(P)$.

To prove the right-to-left direction, we suppose $u \in \beta(P)$. Then u is adjacent to every vertex in P. Then $\beta(u) \supseteq P$. Therefore, $\beta(u)$ is a transaction of DB_G containing P. So, $u \in occ(P)$. This completes the proof. ■

Proposition 3: Given a graph G and a pattern P of DB_G. Then $\beta^G(\beta^G(P)) = CL^{DB_G}(P)$. *Proof:* By Proposition 2, $\beta(\beta(P)) = \beta(occ(P)) = \bigcap_{id(T) \in occ(P)} T = \bigcap_{T \in f(P)} T = g(f(P)) = CL(P)$. ■

Thus, $\beta^G \circ \beta^G$ is a closure operation on patterns of DB_G.

We also prove in the next section that $occ^{DB_G}(P)$ or $\beta^G(P)$ is a closed pattern for any pattern P, and in particular when P is a closed pattern of DB_G. This result together with Propositions 2



A graph G

$\Rightarrow$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	1	0	1	1	1
v_3	1	1	0	1	1
v_4	0	1	1	0	0
v_5	0	1	1	0	0

its adjacency matrix

$\Rightarrow$

$id(T)$	T	items				
		v_1	v_2	v_3	v_4	v_5
v_1	$\beta(v_1)$	0	1	1	0	0
v_2	$\beta(v_2)$	1	0	1	1	1
v_3	$\beta(v_3)$	1	1	0	1	1
v_4	$\beta(v_4)$	0	1	1	0	0
v_5	$\beta(v_5)$	0	1	1	0	0

transformation to DB_G

Fig. 3. A graph G is transformed into a transactional database.

and 3 are critical to relationships between the closed patterns of DB_G and the maximal biclique subgraphs of G.

IV. The Mapping between Maximal Biclique Subgraphs and Closed Pattern Pairs

Lemma 1: Let G be a graph. Let C be a closed pattern of DB_G. Then $f^{DB_G}(occ^{DB_G}(C)) = \{\beta^G(c) \mid c \in C\}$.

Proof: As C is a closed pattern, we have $C = g(f(C))$. It follows easily from the definitions of g and f that $C = \bigcap_{T \in DB_G,\ C \subseteq T} T$. Hence, the set of items in C are exactly those items contained in every transaction $T \in DB_G$ that contains C. Then, by Proposition 2, the set of items in C are exactly those vertices of G that are adjacent to every vertex $id(T)$ where the transaction $T \in DB_G$ contains C. That is, $C = \{c \in V^G \mid c \in \beta(id(T)), T \in DB_G, C \subseteq T\}$. Then it follows from the definition of occurrence set that C is exactly the set of vertices of G that are adjacent to every vertex in $occ(C)$. This implies that $\{\beta(c) \mid c \in C\}$ are exactly those transactions that contain $occ(C)$. In other words, $f(occ(C)) = \{\beta(c) \mid c \in C\}$. ■

Proposition 4: Let G be a graph and C a closed pattern of DB_G. Then $occ^{DB_G}(C)$ is also a closed pattern of DB_G.

Proof: By Lemma 1, $f(occ(C)) = \{\beta(c) \mid c \in C\}$. So, $CL(occ(C)) = g(f(occ(C))) = \bigcap f(occ(C)) = \bigcap_{c \in C} \beta(c) = \beta(C)$. By Proposition 2, $\beta(C) = occ(C)$. Thus $occ(C)$ is a closed pattern. ■

Proposition 5: Let G be a graph and C a closed pattern of DB_G. Then C and its occurrence set has empty intersection. That is, $occ^{DB_G}(C) \cap C = \{\}$.

Proof: Let $v \in occ(C)$. Then v is adjacent to every vertex in C. Since we assume G is a graph without self-loops, $v \notin C$. Therefore, $occ^{DB_G}(C) \cap C = \{\}$. ■

In fact Proposition 5 holds for any pattern P, not necessarily a closed pattern C.

Now, we present the main result of this paper: the pair of a closed pattern C and its occurrence set $occ^{DB_G}(C)$ always yields a distinct maximal biclique subgraph of G.

Theorem 1: Let G be an undirected graph without self-loops. Let C be a closed pattern of DB_G. Then the graph

$$H = \langle C \cup occ^{DB_G}(C), C \times occ^{DB_G}(C)\rangle$$

is a maximal biclique subgraph of G.

Proof: By assumption, C is non-empty and C has a non-zero support in DB_G. Therefore, $occ(C)$ is non-empty. By Proposition 5, $C \cap occ^{DB_G}(C) = \{\}$. Furthermore, for every $v \in occ(C)$, v is adjacent in G to every vertex of C. So, $C \times occ(C) \subseteq E^G$, and every edge of H connects a vertex of C and a vertex of $occ(C)$. Thus, H is a biclique subgraph of G. By Proposition 2, we have $occ^{DB_G}(C) = \beta^G(C)$. By Proposition 3, $C = \beta^G(\beta^G(C))$. By Proposition 2, we derive $C = \beta^G(occ^{DB_G}(C))$. So H is maximal. This finishes the proof. ■

Theorem 2: Let G be an undirected graph without self-loops. Let graph $H = \langle V_1 \cup V_2, E\rangle$ be a maximal biclique subgraph of G. Then, V_1 and V_2 are both closed patterns of DB_G, $occ^{DB_G}(V_1) = V_2$ and $occ^{DB_G}(V_2) = V_1$.

Proof: Since H is a maximal biclique subgraph of G, then $\beta(V_1) = V_2$ and $\beta(V_2) = V_1$. By Proposition 3, $CL(V_1) = \beta(\beta(V_1)) = \beta(V_2) = V_1$. So, V_1 is a closed pattern. Similarly, we can get V_2 is a closed pattern. By Proposition 2, $occ(V_1) = \beta(V_1) = V_2$ and $occ(V_2) = \beta(V_2) = V_1$, as required. ■

Theorems 1 and 2 say that maximal bicliques of G are all in the form of $H = \langle V_1 \cup V_2, E\rangle$, where V_1 and V_2 are both closed patterns of DB_G. Also, for every closed pattern C of DB_G, the graph $H = \langle C \cup occ^{DB_G}(C), C \times occ^{DB_G}(C)\rangle$ is a maximal biclique of G. So, given a graph G, there exists a one-to-one correspondence between the maximal bicliques of G and the closed pattern pairs of DB_G.

Example 4: Consider the graph G and DB_G given in Example 3 again. The adjacency matrix is:

	v_1	v_2	v_3	v_4	v_5
$\beta(v_1)$	0	1	1	0	0
$\beta(v_2)$	1	0	1	1	1
$\beta(v_3)$	1	1	0	1	1
$\beta(v_4)$	0	1	1	0	0
$\beta(v_5)$	0	1	1	0	0

The closed patterns, their support, and their occurrence sets are as follows:

$sup(X)$	closed pattern X	$Y = occ(X)$	$sup(Y)$
3	$\{v_2, v_3\}$	$\{v_1, v_4, v_5\}$	2
4	$\{v_2\}$	$\{v_1, v_3, v_4, v_5\}$	1
4	$\{v_3\}$	$\{v_1, v_2, v_4, v_5\}$	1

This DB_G has a total of 6 closed patterns; and this graph G has exactly three maximal biclique subgraphs:

$$\begin{aligned} H_1 &= \langle\{v_2, v_3\} \cup \{v_1, v_4, v_5\}, \\ &\quad \{(v_2, v_1), (v_2, v_4), (v_2, v_5), (v_3, v_1), (v_3, v_4), (v_3, v_5)\}\rangle \\ H_2 &= \langle\{v_2\} \cup \{v_1, v_3, v_4, v_5\}, \\ &\quad \{(v_2, v_1), (v_2, v_4), (v_2, v_5), (v_3, v_2)\}\rangle \\ H_3 &= \langle\{v_3\} \cup \{v_1, v_2, v_4, v_5\}, \\ &\quad \{(v_3, v_1), (v_3, v_4), (v_3, v_5), (v_3, v_2)\}\rangle \end{aligned}$$

This can be visually verified from Figure 3 easily.

Proposition 6: Let G be a graph. Let C_1 and C_2 be two closed patterns of DB_G. Then $C_1 = C_2$ iff $occ^{DB_G}(C_1) = occ^{DB_G}(C_2)$.

Proof: The proof is obvious, thus omitted. ∎

Propositions 4, 5, and 6 give rise to two corollaries.

Corollary 1: Let G be a graph. Then the number of closed patterns in DB_G is even.

Proof: Suppose there are n closed patterns that appear at least once in DB_G, denoted as C_1, C_2, ..., C_n. As per Proposition 4, $occ(C_1)$, $occ(C_2)$, ..., $occ(C_n)$ are all closed patterns of DB_G. As per Proposition 6, $occ(C_i)$ is different from $occ(C_j)$ iff C_i is different from C_j. So every closed pattern can be paired with a distinct closed pattern by $occ(\cdot)$ in a bijective manner. Furthermore, as per Proposition 5, no closed pattern is paired with itself. This is possible only when the number n is even. ∎

Corollary 2: Let G be a graph. Then the number of closed patterns C, such that both C and $occ^{DB_G}(C)$ appear at least ms times in DB_G, is even.

Proof: As seen from the proof of Corollary 1, every closed pattern C of DB_G can be paired with $occ^{DB_G}(C)$, and the entire set of closed patterns can be partitioned into such pairs. So a pair of closed patterns C and $occ^{DB_G}(C)$ either satisfy or do not satisfy the condition that both C and $occ^{DB_G}(C)$ appear at least ms times in DB_G. Therefore, the number of closed patterns C, satisfying that both C and $occ^{DB_G}(C)$ appear at least ms times in DB_G, is even. ∎

Note that this corollary does not imply the number of frequent closed patterns in DB_G is always even. See the following counter example.

Example 5: Continue with the graph G and DB_G used in Example 4. It is already known that DB_G has a total of 6 closed patterns. But if we set $ms = 3$, then there are only 3 closed patterns—an odd number—that occur at least ms times, viz. $\{v_2, v_3\}$, $\{v_2\}$, and $\{v_3\}$.

V. Efficient Mining of Large Maximal Bicliques

Not all maximal biclique subgraphs are equally interesting. Recall our earlier motivating example involving customers in a mobile communication network. Those two groups of customers with a small size containing only a single person or just a few would be trivial, while if one or both of the groups are large, then they are useful. Hence, we introduce the concept of **large** bipartites to exclude those bipartites with trivial sizes. We also present an efficient algorithm for mining maximal bicliques of large size.

A. Maximal bicliques of large size

Definition 5: A maximal biclique subgraph $H = \langle V_1 \cup V_2, E\rangle$ of a graph G is said to be (p, q)-large if $|V_1|$ or $|V_2|$ is at least p, and the other is at least q.

Proposition 7: Let G be a graph and $C \subset V^G$. Then $H = \langle C \cup occ^{DB_G}(C), C \times occ^{DB_G}(C)\rangle$ is a (p, q)-large maximal biclique subgraph of G, where $p \leq q$ and $|C| \leq |occ^{DB_G}(C)|$, iff C is a closed pattern with $|C| \geq p$ and $sup^{DB_G}(C) \geq q$.

Proof: The proof is easy, and thus omitted. ∎

So, our mining problem in this work can be re-formulated as to enuerate all closed patterns C of DB_G that satisfies $|C| \geq p$ and $sup^{DB_G}(C) \geq q$. It is straightforward for existing closed pattern mining algorithms [42], [43], [38], [34], [39], [36], [35] to get all those closed patterns with the support threshold set as q. However, to move to identify all the (p, q)-large maximal bicliques, these existing algorithms need two extra steps:

1) Remove those frequent closed patterns if their cardinality is less than p (computationally trivial);
2) Pair up the remaining closed patterns to form maximal bicliques (computationally non-trivial).

Furthermore, this filtering approach has two shortcomings: (1) generating many frequent closed pattern with a small cardinality ($< p$) which are useless in constructing large maximal bicliques; (2) unnecessarily listing all closed patterns with the least support of q. The reason is that whenever a closed pattern C is identified, it is the case that $occ^{DB_G}(C)$ is also a closed pattern of DB_G. If $occ^{DB_G}(C)$ has a support $\geq q$, then $occ^{DB_G}(C)$ is redundantly enumerated, resulting in heavy computational redundancy in the existing algorithms to list all (p, q)-large maximal bicliques.

To avoid these two drawbacks, we develop a new algorithm to enumerate only those closed patterns C of DB_G satisfying the following three conditions:

1) $sup^{DB_G}(C) \geq q$,
2) $|C| \geq p$, and
3) $sup^{DB_G}(C) \geq |C|$.

The first two conditions garranttee the completeness of the solution, the third constraint can avoid the computational redundancy. However, we have to note that those closed pattern pairs $|C| = |occ^{DB_G}(C)| \geq q$ are enumerated twice in our algorithm.

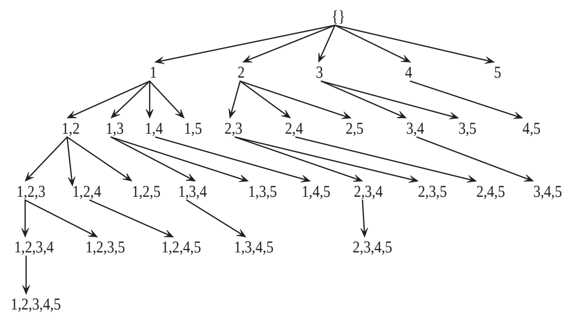


Fig. 4. A set-enumeration tree for a graph with vertex set $\{1, 2, 3, 4, 5\}$.

B. Our algorithm LCM-MBC

Let G be a graph and $I = V^G$ be the set of items of DB_G. As every subset of I can be possibly frequent in DB_G, all of them form the search space of this frequent itemset mining problem. The search space can be represented as a set-enumeration tree [44] as shown in Figure 4. The root of the tree represents the empty set. Each node at level-k represents an itemset containing k items. The subtree rooted at itemset X is called the *sub search space tree* of X. In a search space tree, the items are sorted into some order. For every itemset X in the tree, only items after the last item of X can appear in the sub search space tree of X. This set of items are called *tail items* of X, denoted as $tail(X)$. For example, in Figure 4, all items are sorted into the lexicographic order, so item 4 is in $tail(\{1, 3\})$, but item 2 is not a tail item of $\{1, 3\}$ because item 2 is prior to item 3.

Most of the frequent closed itemset mining algorithms [42], [43], [38], [34], [39], [36], [35] use the depth-first-order to explore the search space, and use the anti-monotone property to prune the search space. The anti-monotone property is stated as follows: if an itemset is not frequent, then none of its supersets is frequent. Based on the anti-monotone property, if an itemset is not frequent, then there is no need to visit the sub search space tree of the itemset. Given two numbers (p, q) ($p \leq q$) as the size constraints on maximal bicliques, we choose the larger one q as the support threshold to maximize the pruning power of the anti-monotone property.

For every itemset X that is visited during the exploration, the tail items of X that are frequent with X are identified, and these items are used to create the child nodes of X. The exploration is then continued on these newly created child nodes. We incorporate two pruning techniques into the depth-first mining framework to prune duplicate and small maximal bicliques. The pseudo-codes of the mining algorithm is shown in Algorithm 1. Algorithm 1 first finds the set of items in $tail(X)$ that are frequent with X (lines 1-3), then uses these items to extend X to obtain longer frequent closed itemsets. The codes at line 10 and 11 are for closed itemset identification, which has been well discussed in frequent closed itemset mining algorithms, so we do not go into the details here. When Algorithm 1 is first called on a graph $G = (V, E)$, X is set to $\{\}$ and $tail(X)$ is set to V.

Algorithm 1 LCM-MBC

```
Input:
    $X$ is an itemset (vertex set)
    $tail(X)$ is the tail items of $X$
    $p$ is the minimum size threshold
    $q$ ($\geq p$) is the minimum support threshold
Description:
 1: for all item $v \in tail(X)$ do
 2:     if $sup^{DB_G}(X \cup \{v\}) < q$ then
 3:         $tail(X) = tail(X) - \{v\}$;
 4: if $|X| + |tail(X)| < p$ then
 5:     return ;
 6: for all item $v \in tail(X)$ do
 7:     $Y = X \cup \{v\}$;
 8:     $tail(Y) = \{u | u \in tail(X) \wedge u\ is\ after\ v\}$;
 9:     if $|Y| + |tail(Y)| \geq p$ then
10:         $Z = Y \cup \{u | u \in tail(Y) \wedge sup^{DB_G}(Y \cup \{u\}) = sup^{DB_G}(Y)\}$;
11:         if $Z$ is a closed itemset then
12:             if $|Z| \geq p$ AND $sup^{DB_G}(Z) \geq |Z|$ then
13:                 Output $Z$ as a closed itemset;
14:             if $sup^{DB_G}(Z) > |Z|$ then
15:                 LCM-MBC($Z$, $tail(Y) - Z$, $p$, $q$);
```

Pruning small maximal biclique subgraphs. To prune small maximal bicliques, we use the size threshold p to prune the search space, in addition to using the support threshold q to prune the search space as explored in common closed itemset mining algorithms. The itemsets appearing in the sub search space of an itemset X are subsets of $X \cup tail(X)$. Therefore, if $|X| + |tail(X)|$ is less than the size threshold p, then there is no need to search in the sub search space tree of X because all the itemsets in that subtree contain less than p items (line 9). Also observe that every itemset Y in the sub search space tree of X with $sup^{DB_G}(Y) \geq q$ must be a subset of $X \cup \{x | x \in tail(X) \wedge sup^{DB_G}(X \cup \{x\}) \geq q\}$. Therefore, if there are less than $(p - |X|)$ items in $tail(X)$ that are frequent with X, then there is no need to search in the subtree of X either (lines 4-5).

Pruning duplicate maximal biclique subgraphs. Given two size thresholds p and q, where $p \leq q$, suppose the larger value q is used as the support threshold and p as the size threshold. Then, a (p, q)-large maximal biclique H with two vertex sets V_1 and V_2 can be in three cases: (i) $|V_1| \geq q$ and $q > |V_2| \geq p$, (ii) $|V_2| \geq q$ and $q > |V_1| \geq p$, and (iii) $|V_1| \geq q$ and $|V_2| \geq q$. In the first case, V_2 is not frequent and only V_1 is frequent, so H is generated exactly once. Similarly in case (ii), H is generated only once. However, H is generated twice in case (iii) where both vertex sets of H are frequent. We prune these duplicate maximal bicliques as follows. If the size of a closed itemset is no less than the support of the closed itemset, then we do not extend the closed itemset further even if the closed itemset is frequent (line 14). Furthermore, a closed itemset is put into the output only if its support is no less than its size (lines 12-13). In this way, all the maximal bicliques in case (iii) whose two vertex sets are of different size are generated exactly once. For maximal bicliques whose two vertex sets are of equal size, we remove the duplicates in a post processing step as follows. Let $H = (V_1, V_2)$ be a maximal biclique whose two vertex sets are of equal size, that is, $|occ^{DB_G}(V_1)| = |V_1|$ and $|occ^{DB_G}(V_2)| = |V_2|$. Two copies of H are generated by Algorithm 1, and they are $H_1 = \langle V_1 \cup occ^{DB_G}(V_1), V_1 \times V_2 \rangle$ and $H_2 = \langle V_2 \cup occ^{DB_G}(V_2), V_1 \times V_2 \rangle$. For each H_i, $i = 1, 2$, we compare its itemset V_i with its occurrence set $occ^{DB_G}(V_i)$ according to the lexicographic order. If V_i is smaller than $occ^{DB_G}(V_i)$, then we keep H_i, otherwise H_i is discarded. The condition that V_i is smaller than $occ^{DB_G}(V_i)$ is true for only one of the two V_is, so one of H_is is kept and the other one is discarded.

The above pruning techniques can be applied to any closed itemset mining algorithm that uses the depth-first-order to explore the search space. In this paper, we choose the state-of-the-art closed itemset mining algorithm LCM [39]. LCM uses a *prefix preserving closure extension* technique to generate all new closed itemsets. This extension technique needs no previously obtained closed itemsets. Hence, the memory usage of LCM does not depend on the number of frequent closed itemsets. The time complexity of LCM is theoretically bounded by a linear function in the number of frequent closed itemsets, that is why LCM is short for Linear time Closed itemset Miner [39]. We incorporate our pruning ideas into LCM specially for mining large maximal bicliques. We call this revised algorithm LCM-MBC, and report its performance in the next section.

Complexity analysis. Let n be the number of vertices in G and m be the number of edges in G. We store the adjancency list of each vertex instead of the adjacency matrix in our computer. As each edge appears twice, the size of our database DB_G is double the number of edges in G, namely the size of DB_G is $2m$. The support of the items in $tail(X)$ can be obtained by scanning the transactions containing itemset X once, so the time complexity of finding items in $tail(X)$ that are frequent with X is $O(m)$ (lines 1-3). Checking whether $|X| + |tail(X)| < q$ requires constant time (lines 4-5). The cost of lines 7, 9, 12 and 14 is also a constant. The cost of line 8 is bounded by the size of $tail(X)$ which in turn is bounded by V^G, so the time complexity of line 8 is $O(n)$. At line 10, itemset Z can be generated by scanning the transactions containing itemset Y once. The set of transactions containing itemset Y is a subset of the transactions in DB_G. Therefore, the total size of the transactions containing itemset Y cannot exceeds the size of DB_G, so the time complexity of line 10 is $O(m)$. At line 11, we need to generate the closure of itemset Z and compare it with Z. If $CL^{DB_G}(Z) = Z$, then itemset Z is a closed itemset, otherwise itemset Z is not a closed itemset. The closure of itemset Z can be generated by intersecting the transactions containing Z, so the time complexity of line 11 is $O(m)$. The cost of outputting an itemset is bounded by the size of the itemset, so the time complexity of line 13 is $O(n)$.

The codes between line 7 and line 14 are called $|tail(X)|$ number of times, and $tail(X)$ is a subset of V^G. Therefore, the cost of lines 6-14 is $O(nm)$. Algorithm 1 is recursively called for every maximal biclique which has a vertex set larger than q (line 15). Therefore, the time complexity of Algorithm 1 is $O(mnN)$, where N is the number of maximal bicliques that have a vertex set larger than q. Put in another way of a general case, let $F(X)$ be the time complexity of finding frequent itemset X and all the frequent supersets of X. Then the recursive formula for the time complexity is:

$$\begin{aligned} F(X) &= O(m) + O(n) * (O(n) + O(m) + O(m) + O(n)) + \\ &\qquad \sum_{v \in tail(X), sup(X \cup \{v\}) \geq q, Z \text{ is closed}} F(X \cup \{v\}) \\ &= O(mn) + \\ &\qquad \sum_{v \in tail(X), sup(X \cup \{v\}) \geq q, Z \text{ is closed}} F(X \cup \{v\}) \end{aligned}$$

where $Z = X \cup \{v\} \cup \{u \mid u \in tail(X)\ AND\ sup(X \cup \{v, u\}) = sup(X \cup \{v\})\}$. The time complexity of Algorithm 1 is $F(\emptyset) = O(mnN)$, where N is the number of frequent closed itemsets.

We need an extra step to generate the occurrence set of each closed itemset. The time complexity of generating one occurrence set is $O(m)$. After the occurrence sets are generated, we remove duplicate maximal bicliques whose two vertices are of equal size. This is done by comparing the two vertex sets of every generated maximal biclique, so the time complexity of removing duplicates is $O(nN)$. In summary, the total time complexity is still $O(mnN)$.

The space complexity of Algorithm 1 depends on the implementation details. It is proportional to the graph size, but is irrelevant to the number of maximal bicliques generated because we do not need to store maximal bicliques during the mining process. In particular, the LCM algorithm needs to maintain the database in memory, and it also maintains the occurrence sets of the itemsets on the current exploration path. The depth of the path is bounded by the number of items n. At each level, the occurrence sets of the itemsets that are of the same parent are maintained, and their total size is bounded by the size of the database m. Therefore, the space complexity of LCM-MBC is $O(nm)$.

Theorem 3: Given a graph G and two size thresholds p and q, Algorithm 1 discovers the complete set of (p, q)-large maximal bicliques in G.

Proof: The correctness and completeness of Algorithm 1 is guaranteed by Proposition 7. ■

VI. Experimental Results

We evaluate the efficiency of our LCM-BMC algorithm by comparing it to the consensus algorithm MICA [28] and the LCM algorithm [39]. Our experiments are conducted on both benchmark and random graph datasets. We also apply our method to a large protein interaction network to find interacting protein groups, and report interesting results related to bioinformatics. Our machine is a PC with a CPU clock rate 3.2GHz and 2GB of memory. The implementation of our LCM-MBC algorithm is modified from the source codes of the LCM algorithm by incorporating the two pruning techniques. The source codes of LCM were download at `http://fimi.cs.helsinki.fi/src/`.

A. *Efficiency comparison*

As the LCM algorithm [39] outputs only individual closed itemsets, we added a post-processing step to LCM to pair up these closed itemsets such that every pair can form a maximal biclique. This ensures a fair comparison, as our algorithm produces maximal bicliques. The MICA [28] algorithm outputs exactly what LCM-MBC outputs. The main idea of MICA is based on the observation that if $B_1 = (X_1, Y_1)$ and $B_2 = (X_2, Y_2)$ are two bicliques, then $(X_1 \cup X_2, Y_1 \cap Y_2)$, $(X_1 \cap X_2, Y_1 \cup Y_2)$, $(X_1 \cup Y_2, Y_1 \cap X_2)$ and $(X_1 \cap Y_2, Y_1 \cup X_2)$, called the *consensuses* of B_1 and B_2, are also bicliques. The MICA algorithm starts with a collection of biclique subgraphs C which covers the edge set of a graph G, and applies repeatedly two transformations—the absorption and the consensus adjunction—and stops when neither of the transformations can be applied. In the absorption step, if a complete bipartite B_1 is a subgraph of another biclique B_2, then B_1 is removed from C. In the consensus adjunction step, if any of the consensuses of two bicliques is not a subgraph of a biclique in C, then the consensus is added to C. The time complexity of the MICA algorithm is $O(n^3 \cdot N)$, where n is the number of vertices and N is the number of maximal biclique subgraphs. Some optimization techniques have been adopted to improve the performance of MICA. We downloaded the source codes of MICA from `http://genome.cs.iastate.edu/supertree/download/biclique/README.html`.

Table I shows the performance of the three algorithms on randomly generated graphs. (A time limit was set on the running time of the three algorithms. If the running time of an algorithm exceeds one hour, we terminate the program.) The first two columns of Table I are the number of vertices and edges in the graphs. The edge density of a graph is the ratio of the number of edges in the graph to the total possible number of edges in the graph. Columns 5, 6 and 7 show the average total running time of the three algorithms over 10 runs. The running time includes both CPU time and I/O time. On all the graphs, we set $p = q = 1$. Both LCM and LCM-MBC perform significantly better than MICA. It indicates that frequent closed itemset mining techniques are much more efficient than the consensus algorithms for mining maximal bicliques. With $p = q = 1$, all the maximal bicliques are large bicliques. In this case, LCM-MBC cannot use the size constraint to prune the search space, but it can prune duplicate maximal bicliques. Therefore, LCM-MBC can still run faster than LCM even when $p = q = 1$. Note that LCM-MBC did not double the speed of LCM because both of them share the same large amount of tree-construction and I/O time.

The last three columns shows the average delay per maximal biclique of the three algorithms. The average delay of LCM and LCM-MBC is more stable than MICA with the increase of edge density and the number of vertices, which indicates that the time complexity of LCM and LCM-MBC is lower than MICA.

Table II shows six datasets obtained from the Second DIMACS Challenge benchmarks[3] and a protein-protein interaction data of yeast where vertices are proteins and edges are interactions between proteins. The protein-protein interaction dataset is downloaded from `http://dip.doe-mbi.ucla.edu/`. Figure 5 shows the running time of the three algorithms on these datasets. On all the datasets, we set the two size thresholds to the same

[3]`ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`

TABLE I

RUNNING TIME ON RANDOM GRAPHS. AT EACH ROW OF THE TABLE, THE PERFORMANCE WAS AVERAGED OVER 10 RANDOMLY GENERATED GRAPHS OF THE SAME VERTEX AND EDGE NUMBER.

		edge	#maximal	total running time (sec)			average delay per max biclique (ms)		
#vertices	#edges	density	bicliques	MICA	LCM	LCM-MBC	MICA	LCM	LCM-MBC
100	495	0.10	615	0.063	0.012	0.010	0.102	0.0195	0.0163
100	990	0.20	5686	0.919	0.080	0.057	0.164	0.0141	0.0100
100	1485	0.30	44258	13.128	0.707	0.472	0.292	0.0160	0.0107
100	1980	0.40	398794	200.489	6.680	4.452	0.508	0.0168	0.0112
100	2475	0.50	4803620	>1 hour	89.861	60.966	-	0.0187	0.0127
500	2494	0.02	1443	0.311	0.039	0.025	0.209	0.0270	0.0173
500	6237	0.05	20387	9.662	0.322	0.222	0.472	0.0158	0.0109
500	12475	0.10	213808	233.644	6.018	4.322	1.09	0.0281	0.0202
500	18712	0.15	2112584	>1 hour	70.104	45.289	-	0.0332	0.0214
1000	4994	0.01	2084	0.790	0.039	0.035	0.378	0.0187	0.0168
2000	19989	0.01	17669	21.956	0.376	0.311	1.248	0.0213	0.0176
4000	79979	0.01	211081	829.001	5.976	4.684	3.933	0.0283	0.0222
8000	319959	0.01	2551816	>1 hour	126.294	91.743	-	0.0495	0.0360

TABLE II

A PROTEIN INTERACTION GRAPH AND 5 DIMACS BENCHMARK GRAPHS

Datasets	#vertices	#edges	edge density
yeast-p2p	4904	17440	0.00145
c-fat500-1	500	4459	0.0357
johnson16-2-4	120	5460	0.765
keller4	171	9435	0.649
c-fat200-2	200	3235	0.163
johnson8-4-4	70	1855	0.768

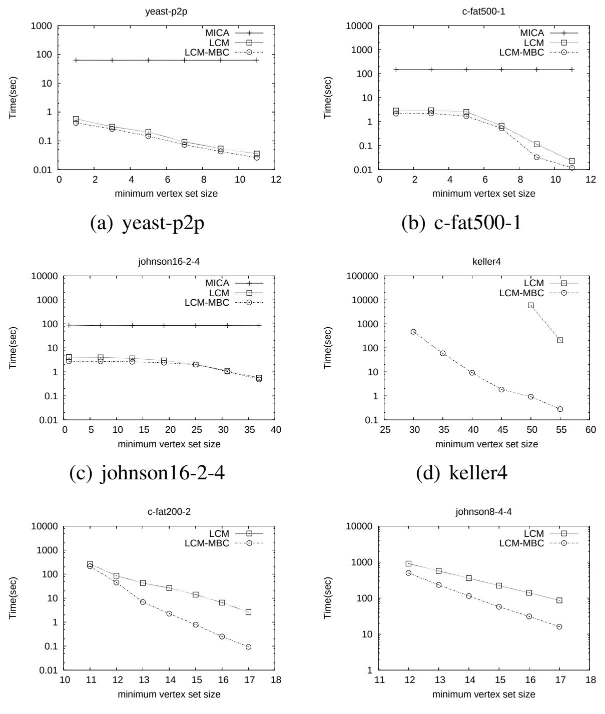
value, that is, $p = q$.

The MICA algorithm was able to complete on only the first three datasets which have a relatively small number of maximal biclique subgraphs, but MICA terminated abnormally on the other datasets which have a prolific number of maximal bicliques. The reason of the abnormal termination is that MICA has to keep all the maximal bicliques in the memory during the mining process, so if the number of maximal biclique exceeds the memory, the program collapses. MICA does not use the size constraints to prune the search space. It mines all maximal biclique subgraphs, and then filters out small maximal bicliques in the output step. Therefore, the running time of MICA is almost constant with respect to the size constraints, and the gap between MICA and LCM, LCM-MBC increases with the increase of the size threshold.

Algorithms LCM and LCM-MBC show similar performance on the first three datasets, where the number of maximal bicliques is small. But, the LCM-MBC algorithm is several times faster than LCM on the other three datasets, where the number of maximal bicliques is very large. The gap between LCM and LCM-MBC increases with the increase of the size threshold, it indicates that LCM-MBC can effectively use the size threshold to prune small and duplicate bicliques during the mining process.

B. A real life application

The topological sub-structures of protein interaction networks such as *protein hubs*, protein *k-cores*/quasi-cliques, and protein *quasi-bipartites* are extensively studied in the bioinformatics field [21], [45]. A protein hub is a protein that interacts with many other proteins; a protein k-core is a set of proteins in which every protein is interacting with at least k other proteins in the set. To identify all protein hubs, only one scan of the network



(a) yeast-p2p (b) c-fat500-1

(c) johnson16-2-4 (d) keller4

(e) c-fat200-2 (f) johnson8-4-4

Fig. 5. Running time comparison on benchmark graph datasets. The MICA algorithm terminated abnormally on the keller4, c-fat200-2, and johnson8-4-4 datasets because of a memory problem.

is needed. The discovery of all k-cores is a bit complicated. A recent algorithm proposed by Pei et al [13] can be used for the efficient enumeration of k-cores (where quasi-clique is called).

Here we demonstrate the speed of our algorithm LCM-MBC for listing all maximal biclique subgraphs from a protein interaction network. As there are many physical protein interaction networks corresponding to different species, we take the most comprehensive yeast physical and genetic interaction network [46] as an example. This graph consists of 4904 vertices and 17440 edges, after removing 185 self loops and 1413 redundant edges from the original 19038 interactions, as shown in Table II. The average size of the neighborhood of a protein is 3.56.

Some interesting results are reported in Table III, where the second column shows the total number of frequent closed patterns

TABLE III
CLOSED PATTERNS IN A YEAST PROTEIN INTERACTION NETWORK.

support threshold (ms)	# of frequent closed patterns	# of qualified closed patterns	time in sec.
1	121314	121314	0.59
2	117895	114554	0.50
3	105854	95920	0.44
4	94781	80306	0.40
5	81708	60038	0.36
6	66429	36478	0.32
7	50506	15800	0.25
8	36223	3716	0.21
9	25147	406	0.11
10	17426	34	0.07
11	12402	2	0.06
12	9138	0	0.05

whose support level is at least the threshold number in the column one. The third column of this table shows the number of closed patterns whose cardinality and support are both at least the support threshold; all such closed patterns are termed **qualified** closed patterns. Only these qualified closed patterns can be used to form maximal biclique subgraphs $H = \langle V_1 \cup V_2, E\rangle$ such that both of $|V_1|$ and $|V_2|$ meet the thresholds, namely those (ms, ms)-large maximal biclique subgraphs. From the table,

- We can re-confirm that the number of all closed patterns—corresponding to those with the support threshold of 1—is even. Moreover, the number of qualified close patterns with cardinality no less than any other support level is also even, as expected from Corollary 2.
- The algorithm runs fast. The program could complete within one second for all situations reported here. This indicates that enumerating all maximal biclique subgraphs from a large protein interaction network can be practically solved by using algorithms for mining closed patterns. The speed is hundreds of times faster than the MICA algorithm.
- A so-called “many-few” property [45] of protein interactions is observed again in our experiment results. The “many-few” property says that: a protein hub tends *not* to interact with another protein hub [45]. In other words, highly connected proteins are separated by low-connected proteins. This is most clearly seen in Table III at the higher support thresholds. For example, at the support threshold 11, there are 12402 protein groups that have full interactions with at least 11 proteins. But there are only two groups, as seen in the third column of the table, that each contain at least 11 proteins and that have full mutual interaction.

VII. ROLE OF SELF-LOOPS

So far, all graphs are assumed to be undirected and without self-loops. If self-loops are allowed, then the main diagonal of the adjacency matrix of a graph are not always 0’s. Then the number of closed patterns in the adjacency matrix is not necessarily an even number. This is a property different from Corollary 1.

Example 6: Let G be a clique graph with n vertices $v_1, v_2, \ldots, v_n$. Suppose every vertex of G is self-connected. Then, the adjacency matrix of this graph is an n-squared matrix with 1’s everywhere. This adjacency matrix has one and only one closed pattern which is $\{v_1, v_2, \ldots, v_n\}$ with a support of n. Thus the number of closed patterns in this graph is odd. However, if remove all the self-loops, the number of maximal biclique subgraphs of the resulting graph can be proved to be $(2^n - 2)/2$, while the number of closed patterns to be $2^n - 2$.

This change in the number of closed patterns does not mean the number of maximal biclique subgraphs is changed. In fact, the two graphs (with or without self-loops) have the same number of maximal biclique subgraphs.

Definition 6: Let G be a graph. Its self-cleared graph, denoted G^*, is defined as $G^* = \langle V^G, E^G - \{(x,x) \mid x \in V^G\}\rangle$.

Theorem 4: Let G be a connected undirected graph where selp loops are allowed. Let C be a closed pattern of DB_{G^*}. Then the graph

$$H = \langle C \cup occ^{DB_{G^*}}(C), C \times occ^{DB_{G^*}}(C)\rangle$$

is a maximal biclique subgraph of G.

Proof: By Theorem 1, we conclude that H is a maximal biclique subgraph of G^*. Therefore, $C, occ^{DB_{G^*}}(C) \subset V^{G^*} = V^G$, and $C \times occ^{DB_{G^*}}(C) \subseteq E^{G^*} \subseteq E^G$. Thus, H is a biclique subgraph of G. If $C \cup occ^{DB_{G^*}}(C) = V^G$, then H is a maximal biclique subgraph of G. We next prove H of the other case is also a maximal biclique of G by contradiction. Let $x \in V^G$ but $x \notin C, occ^{DB_{G^*}}(C)$. Assume $H' = \langle (C \cup x) \cup occ^{DB_{G^*}}(C), (C \cup x) \times occ^{DB_{G^*}}(C)\rangle$ is a biclique in G. Then, x is adjacent to every vertex in $occ^{DB_{G^*}}(C)$. Thus, H' is a biclique of G^*. However, H is a maximal biclique of G^*. Here is a contradiction. Therefore, H is a maximal biclique of G. ■

Theorem 5: Let G be a connected undirected graph where self loops are allowed. Let graph $H = \langle V_1 \cup V_2, E\rangle$ be a maximal biclique subgraph of G. Then, V_1 and V_2 are both closed patterns of DB_{G^*}, and $occ^{DB_{G^*}}(V_1) = V_2$ and $occ^{DB_{G^*}}(V_2) = V_1$.

Proof: It is obvious that H is also a maximal complete bipartite subgraph of G^*. Then the theorem follows immediately from Theorem 2. ■

The above two theorems say that the maximal biclique subgraphs of any graph G, even with self loops, can be determined by the closed patterns of the adjacency matrix of G^*.

VIII. RELATION WITH FORMAL CONCEPT ANALYSIS, AND BICLUSTERING

In contrast to the current work which transforms a general graph into a special transactional database, Zaki and Ogihara had an innovative work [33] on how a transactional database can be converted into a bipartite graph for itemset discovery. They have also studied how association rule mining is closely related to Formal Concept Analysis (FCA) [40], a powerful knowledge representation tool. In the following, we give a short review for FCA, and then explain why we did not choose FCA to model and discover maximal bicliques from a large graph.

A *formal context* is a triple (O, A, R), where O is a set of objects, and A a set of attributes, R a binary relation between O and A, i.e. $R \subseteq O \times A$. A *formal concept* of a formal context (O, A, R) is defined as a pair (X, Y) with $X \subseteq O$, $Y \subseteq A$ such that (X, Y) is maximal with the property $X \times Y \subseteq R$. That is, $R(X) = Y$, $R(Y) = X$, where $R(X) = \{a \mid (o,a) \in R, \text{for all } o \in X\}$, $R(Y) = \{o \mid (o,a) \in R, \text{for all } a \in Y\}$.

Given a graph $H = (V, E)$, to model H by a formal context (O, A, R), we can set $O = A = V$, and let R defined by the adjacency matrix of H. Now the question is that whether a formal concept (X, Y) of the formal context (V, V, R) is always a maximal biclique of H. The answer is no. Because $R(X) \cap R(Y)$

may be non-empty, e.g. when H contain some self-loops. Even H is assumed to be a graph without self-loops, it is not obvious that $R(X) \cap R(Y) = \emptyset$. However, this is a necessary requirement by a biclique—the two vertex sets must be non-overlapping. Therefore, it is not effective for a formal concept to model a maximal biclique unless a restrictive constraint is added.

Even if a maximal clique is modeled by a formal concept by imposing the constraint that H has to be a self-loopless graph, this model could not provide any help to claim that the mining of maximal bicliques from a graph H is equivalent to the mining of closed patterns of the adjacency matrix of H. Therefore, for mining maximal bicliques as shown in this work, the FCA model is not a necessary tool. The necessary thing is to prove that $\beta(P)$ and $\beta(\beta(P))$ are a pair of closed patterns of the adjacency matrix, and thus to achieve the main objective of this work—efficiently enumerating large maximal bicliques from a graph through closed patterns of the adjacency matrix.

The enumeration of maximal bicliques from a graph, i.e. the mining of closed pattern pairs from the adjacency matrix is a special case of the biclustering problem [41]. Given a matrix $\mathbf{B}$ with n rows and m columns, where every element b_{ij} is a real number, a bicluster of $\mathbf{B}$ refers to a sub-matrix of $\mathbf{B}$ such that the elements in this sub-matrix satisfy some specific characteristics of homogeneity; while biclustering refers to the problem of mining a set of biclusters from $\mathbf{B}$. The biclustering problem is more general than the biclique enumeration problem because the former requires just an $n \times m$ matrix with real numbers as its elements, not necessarily a squared binary matrix as required by the latter. Also, a biclique corresponds to a sub-matrix full of the constant 1, but the sub-matrix corresponding to a bicluster is just required to satisfy some homogeneity property such as coherence, constant scaling and shift, small mean squared residues, etc [41]. However, some real-life applications such as those mentioned in Introduction just need the maximal biclique results, instead of the biclusters. Therefore, algorithms tailoring for efficient enumeration of maximal bicliques is still an interesting and important problem in discrete mathematics.

IX. Conclusion

In this paper, we have studied the problem of listing all maximal biclique subgraphs from a graph that does not have any self-loops. We have proved that this problem is equivalent to the mining of closed patterns from the adjacency matrix of this graph. For a graph where self-loops are allowed, we have also proved that its maximal biclique subgraphs can be determined by the closed patterns from the adjacency matrix of this graph after removing its self-loops. So, the maximal biclique subgraphs of every graph, even with self-loops, can be enumerated by using closed pattern mining algorithms. However, direct use of these algorithms causes a duplicated enumeration. So, we have proposed a new method, called LCM-MBC, which can avoid the duplicated enumeration and which can prune small bipartite subgraphs efficiently. Experimental results on many random graphs and 6 benchmark graph datasets have shown that our LCM-MBC algorithm can be significantly faster than MICA and LCM for enumerating large bipartite subgraphs from dense graphs with many vertices. In this paper, we have also reported some bipartite results obtained from a large protein interaction network and found these results can reconfirm the so called “many-few” property of protein interactions. Our algorithm is very fast (within one second) for mining all the interacting protein groups. This indicates that our method have good potential in real-life applications in the web mining, communication systems, and biological fields, where data are large and timing requirement is critical. An upper bound (i.e. $2^{n-1} - 1$) and a lower bound (i.e. 1) on the number of maximal biclique subgraphs are briefly mentioned in this paper. It is an open question that whether there exist a family of connected graphs all having exactly q closed patterns, where q is a pre-specified even number between 2 and $2^n - 2$. This is also one of our future work.

Acknowledgement

We are thankful to three anonymous reviewers from the Program Committee of PKDD 2005 for their useful comments and suggestions on an early conference version of this paper, and also grateful to the two TKDE anonymous reviewers for their excellent comments.

References

[1] A. Inokuchi, T. Washio, and H. Motoda, “Complete mining of frequent patterns from graphs: Mining graph data,” *Machine Learning*, vol. 50(3), pp. 321–354, 2003.

[2] M. Kuramochi and G. Karypis, “An efficient algorithm for discovering frequent subgraphs,” *IEEE Transaction on Knowledge and Data Engineering*, vol. 16(9), pp. 1038–1051, 2004.

[3] M. Koyutrk, A. Grama, and W. Szpankowski, “An efficient algorithm for detecting frequent subgraphs in biological networks,” *Bioinformatics*, vol. Supp. 1, pp. 200–207, 2004.

[4] M. Kuramochi and G. Karypis, “Grew-a scalable frequent subgraph discovery algorithm,” in *ICDM*, 2004, pp. 439–442.

[5] J. Huan, W. Wang, and J. Prins, “Efficient mining of frequent subgraphs in the presence of isomorphism,” in *ICDM*, 2003, pp. 549–552.

[6] X. Yan and J. Han, “gspan: Graph-based substructure pattern mining,” in *ICDM*, 2002, pp. 721–724.

[7] J. Huan, W. Wang, J. Prins, and J. Yang, “Spin: mining maximal frequent subgraphs from graph databases,” in *KDD*, 2004, pp. 581–586.

[8] A. O. Mendelzon and P. T. Wood, “Finding regular simple paths in graph databases,” *SIAM J. Computing*, vol. 24(6), pp. 1235–1258, 1995.

[9] X. Yan, P. S. Yu, and J. Han, “Substructure similarity search in graph databases,” in *SIGMOD Conference*, 2005, pp. 766–777.

[10] C. Wang, W. Wang, J. Pei, Y. Zhu, and B. Shi, “Scalable mining of large disk-based graph databases,” in *KDD*, 2004, pp. 316–325.

[11] S. Flesca and S. Greco, “Querying graph databases,” in *EDBT*, 2000, pp. 510–524.

[12] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou, “Mining coherent dense subgraphs across massive biological networks for functional discovery,” *Bioinformatics*, vol. Supp. 1, pp. 213–221, 2005.

[13] J. Pei, D. Jiang, and A. Zhang, “On mining cross-graph quasi-cliques,” in *KDD*, 2005, pp. 228–238.

[14] X. Yan, X. J. Zhou, and J. Han, “Mining closed relational graphs with connectivity constraints,” in *KDD*, 2005, pp. 324–333.

[15] X. Yan and J. Han, “Closegraph: mining closed frequent graph patterns,” in *KDD*, 2003, pp. 286–295.

[16] T. Washio and H. Motoda, “State of the art of graph-based data mining,” *SIGKDD Explorations*, vol. 5(1), pp. 59–68, 2003.

[17] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener, “Graph structure in the web,” *Computer Networks*, vol. 33(1-6), pp. 309–320, 2000.

[18] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, “Trawling the web for emerging cyber-communities,” *Computer Networks*, vol. 31(11-16), pp. 1481–1493, 1999.

[19] T. Murata, “Discovery of user communities from web audience measurement data,” in *Proceedings of The 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004)*, 2004, pp. 673–676.

[20] J. E. Rome and R. M. Haralick, “Towards a formal concept analysis approach to exploring communities on the world wide web,” in *International Conference on Formal Concept Analysis*, 2005, pp. 33–48.

[21] 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 proteinprotein interaction network in budding yeast,” *Nucleic Acids Research*, vol. 31 (9), pp. 2443–2450, 2003.

[22] H. Li, J. Li, and L. Wong, “Discovering motif pairs at interaction sites from protein sequences on a proteome-wide scale,” *Bioinformatics*, vol. 22, pp. 989–996, 2006.

[23] D. J. Reiss and B. Schwikowski, “Predicting protein-peptide interactions via a network-based motif sampler,” *Bioinformatics*, vol. 20 (suppl.), pp. i274–i282, 2004.

[24] A. H. Tong, B. Drees, G. Nardelli, G. D. Bader, B. Brannetti, L. Castagnoli, M. Evangelista, S. Ferracuti, B. Nelson, S. Paoluzi, M. Quondam, A. Zucconi, C. W. Hogue, S. Fields, C. Boone, and G. Cesareni, “A combined experimental and computational strategy to define protein interaction networks for peptide recognition modules,” *Science*, vol. 295, pp. 321–324, 2002.

[25] A. C. Driskell, C. Ane, J. G. B. M. M. McMahon, B. C. OMeara, and M. J. Sanderson, “Prospects for building the tree of life from large sequence databases,” *Science*, vol. 306, pp. 1172–1174, 2004.

[26] 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*, vol. 20(7), pp. 1036–1042, 2003.

[27] C. Yan, J. G. Burleigh, and O. Eulenstein, “Identifying optimal incomplete phylogenetic data sets from sequence databases,” *Molecular Phylogenetics and Evolution*, vol. 35(3), pp. 528–535, 2005.

[28] 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*, vol. 145(1), pp. 11–21, 2004.

[29] D. Eppstein, “Arboricity and bipartite subgraph listing algorithms,” *Information Processing Letters*, vol. 51, pp. 207–211, 1994.

[30] 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.

[31] V. M. Dias, C. M. de Figueiredo, and J. L. Szwarcfiter, “Generating bicliques of a graph in lexicographic order,” *Theoretical Computer Science*, vol. 337, pp. 240–248, 2005.

[32] 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, D.C.: ACM Press, May 1993, pp. 207–216.

[33] M. J. Zaki and M. Ogihara, “Theoretical foundations of association rules,” in *Proc. 3rd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998.

[34] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules,” in *Proceedings of the 7th International Conference on Database Theory (ICDT)*, 1999, pp. 398–416.

[35] M. J. Zaki and C.-J. Hsiao, “CHARM: An efficient algorithm for closed itemset mining,” in *Proceedings of the Second SIAM International Conference on Data Mining*, 2002.

[36] J. Wang, J. Han, and J. Pei, “CLOSET+: Searching for the best strategies for mining frequent closed itemsets,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’03), Washington, DC, USA*, 2003, pp. 236–245.

[37] G. Grahne and J. Zhu, “Efficiently using prefix-trees in mining frequent itemsets,” in *Proceedings of FIMI’03: Workshop on Frequent Itemset Mining Implementations*, 2003.

[38] ——, “Fast algorithms for frequent itemset mining using fp-trees,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347–1362, October 2005.

[39] T. Uno, M. Kiyomi, and H. Arimura, “LCM ver.2: Efficient mining algorithms for frequent/closed/maximal itemsets,” in *IEEE ICDM’04 Workshop FIMI’04 (International Conference on Data Mining, Frequent Itemset Mining Implementations)*, 2004.

[40] G. Stumme, R. Wille, and U. Wille, “Conceptual knowledge discovery in databases using formal concept analysis methods,” in *European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD)*, 1998, pp. 450–458.

[41] S. C. Madeira and A. L. Oliveira, “Biclustering algorithms for biological data analysis: a survey,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1(1), pp. 24–45, 2004.

[42] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal, “Mining minimal non-redundant association rules using frequent closed itemsets,” *Computational Logic*, pp. 972–986, 2000.

[43] B. Goethals and M. J. Zaki, “FIMI’03: Workshop on frequent itemset mining implementations,” in *Third IEEE International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations*, 2003, pp. 1–13.

[44] R. Rymon, “Search through systematic set enumeration,” in *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge MA, October 1992, pp. 539–550.

[45] S. Maslov and K. Sneppen, “Specificity and stability in topology of protein networks,” *Science*, vol. 296, pp. 910–913, 2002.

[46] B. J. Breitkreutz, C. Stark, and M. Tyers, “The grid: The general repository for interaction datasets,” *Genome Biology*, vol. 4, no. 3, p. R23, 2003.

PLACE PHOTO HERE

Jinyan Li is an Associate Professor at the School of Computer Engineering, Nanyang Technological University. He received his PhD in computer science from the University of Melbourne in 2001. He had 70 articles published in the fields of bioinformatics, data mining, and machine learning. His research focuse is currently on protein interaction networks in computational biology, mining of statistically important discriminative patterns, mining of interaction subgraphs, and classification methods. One of his most interesting work is bioinformatics research on infectious diseases in collaboration with a biological group from MIT.

PLACE PHOTO HERE

Guimei Liu is a post-doc research fellow at School of Computing, National University of Singapore. She received her PhD in computer science from Hong Kong University of Science and Technology in 2005. Her current research interests include frequent pattern mining and its applications, and protein interaction networks mining and analysis.

PLACE PHOTO HERE

Haiquan Li received his BEng and MEng in computer science from Huazhong University of Science and Technology and his PhD in computer science from National University of Singapore. He is a Research Associate of The Samuel Roberts Noble Foundation. His research is on data mining algorithms for bioinformatics applications, such as itemset mining, graph mining and classification for protein interactome and plant genomics as well as prediction and characterization of membrane transporters from primary sequences.

PLACE PHOTO HERE

Limsoon Wong is a professor of computer science and pathology at The National University of Singapore. He currently works mostly on knowledge discovery technologies and their application to biomedicine. He has also done significant research in database query language theory and finite model theory, as well as significant development work in broad-scale data integration systems. A few of his papers are among the best cited of their respective fields. He was recently an architect of Singapore Science and Technology Plan 2010, chairing the section on information technology and multimedia. In recognition of his contributions to science and technology, he has received several international and national awards. He serves on the editorial boards of Journal of Bioinformatics and Computational Biology (ICP), Bioinformatics (OUP), and Drug Discovery Today (Elsevier). He is a scientific advisor to GeneticXchange (USA), Molecular Connections (India), CellSafe International (Malaysia), and KooPrime (Singapore). He received his BSc(Eng) from Imperial College London and his PhD from The University of Pennsylvania.