To the Graduate Council:

I am submitting herewith a dissertation written by Yun Zhang entitled "Scalable graph algorithms with applications in genetics." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michael A. Langston

Major Professor

We have read this dissertation
and recommend its acceptance:

Dr. Michael W. Berry

Dr. Elissa J. Chesler

Dr. Lynne E. Parker

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Scalable Graph Algorithms with Applications in Genetics

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Yun Zhang
December 2008

Dedication

This dissertation is dedicated to my parents, Youlin Liu and Minhua Zhang, and my husband Huadong Liu, who have always been there for me.

Acknowledgments

First, I would like to thank my academic and research advisor Dr. Michael A. Langston, for introducing me to the area of computational biology and for encouraging me to work in this area. I sincerely appreciate his expert guidance and consistent support. Throughout my studies he encouraged me to develop independent thinking and research skills. Second, I would like to thank my research supervisor Dr. Elissa J. Chesler for having patience in mentoring me to understand the biological meanings of the computational results and for always giving me encouragement and support. I would also like to express my gratitude to my other exceptional doctoral committee members, Dr. Lynne E. Parker and Dr. Michael W. Berry for their time and input to this dissertation.

Being a member of the Graph Theory group at the University of Tennessee, Knoxville, and the Systems Genetics group at Oak Ridge National Laboratory, I owe many thanks to both staff and students. I would like to give special thanks to Dr. Faisal N. Abu-Khzam, Dr. Henry Suters, and Dr. Nicole Baldwin, who gave me great help during my first semesters in the team. I would like to thank Dr. Nagiza Samatova for her guidance and support for two years. I would like to give thanks to Dr. Brynn H. Voy and Dr. Arnold Saxton for their advice in biology and statistics. I would like to thank Suzanne Baktash for her help on my dissertation. I would also like to give my thanks to Dr. Roumyana Kirova for helping me with techniques in statistics. I enjoyed working with Dr. Xinxia Peng, Dr. Lan Lin, Jon Scharff, John Eblen, Dr. Andy Perkins, Gary Rodgers, Jeremy Jay, Charles Philips, Vivek Philip, Bhavesh Borate, and Sudhir Naswa. I would like to thank them for the many conversations that have inspired my work.

I have been fortunate to meet many friends outside of my study. This dissertation could not have been written without their encouragement and support. The friendship with Xingyan Li, Yuanyuan Li, Siyuan Liu, Fang Tang, Yifan Tang, Chang Cheng, Teng Ma and many others have been the source of the greatest pleasures.

Finally I would like to give my heartfelt thanks to my family for providing an enormous support. My parents, Youlin Liu and Minhua Zhang were always there for me, guiding me to the right direction and supporting my decisions. Together with my mother-in-law, Yijuan Tao, they helped me finish the dissertation on time by providing love and care for my son. The largest thank you goes to my devoted husband, Huadong Liu, who deserves unique recognition for being incredibly understanding, supportive and, most of all, patient during the course of this dissertation. I also thank my new son Erick Liu, whose good sleeping habits helped make my work possible.

Lastly, I appreciate the financial support from DOE, NIH, NSF, and Jackson Laboratory that funded the research discussed in this dissertation.

Abstract

Graph theoretical approaches have been widely used to solve problems arising in bioinformatics and genomic analysis. In particular, enumeration problems such as maximal clique and maximal biclique finding are cores for addressing many biological problems, such as the integration of genome mapping data. However, the enumeration problems may become computation and memory bottlenecks for genome-scale elucidation of biological networks due to their $\mathcal{NP}$-hard nature and the huge memory requirements.

Therefore, this research is interested in developing exact, scalable, and efficient algorithms for these biological problems. The *Clique Enumerator* including a maximal clique enumeration algorithm and its parallel implementation is developed to provide scalable and exact solutions to the maximal clique problem by taking advantage of (a) ultra-large globally addressable memory architectures, (b) the theory of fixed parameter tractability, and (c) a novel bitmap data representation scheme for memory and search space reduction. An implementation of the parallel algorithm scales linearly to at least 64 processors.

Maximal biclique finding on bipartite graphs is the other focus of this research because of its key role in the study of gene-phenotype association graphs built from heterogeneous data types. A general purpose algorithm is developed for enumerating maximal bicliques in a bipartite graph, without any problem restriction. Experimental results using both gene expression data and synthetic data indicate that the new algorithm can be two to three orders of magnitude faster than the best previous alternatives.

The clique and biclique enumeration algorithms are applied to two applications in genetics: the linkage disequilibrium analysis in population genetics with clique algorithms, and the ontology discovery for systems genetics with biclique algorithms. The clique model is applied to extract linkage disequilibrium networks from all loci on the whole-genome. The structures of such networks are analyzed to compare the genetic structure of populations. The biclique model is applied to enumerate bicliques in gene-phenotype association graphs constructed from empirical gene sets. Bicliques are integrated into a directed acyclic graph to provide an empirical discovery of phenotype ontology.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in biochemistry and genome studies have opened new opportunities for interdisciplinary research related to biotechnology. The advent of microarrays and other high-throughput biological technologies has led to the rapid generation of immense data sets involving genes and gene products, which has presented a number of computational challenges. Some of these challenges can be handled using combinatorial algorithms. In particular, many problems arising in bioinformatics and genomic analysis can be formulated in terms of optimization or enumeration problems in specifically constructed graphs.

Graph-theoretical methods sometimes become computational bottlenecks for large-scale biological network analysis. This is because our current understanding of the $\mathcal{NP}$-hard nature of these problems translates to exponential run times. Therefore, these approaches tend not to scale as the problem size grows. Approximate solutions with polynomial run times have been proposed to save computation time, but exact solutions are widely preferred due to the time and expense involved in biological data capture. For example, recombinant inbred strains take more than eight years to become isogenetically pure; pathway/network elucidation of yeast requires 45,000 synthetic genetic arrays; and Affymetrix U133 arrays contain more than 30,000 probesets. Due to the high value of the underlying biological data, approximation algorithms are often unacceptable. Therefore, efficient and scalable optimization methods are crucial to systems-level biological applications.

Graph-theoretical approaches, especially those characterized as enumerative, are often memory-intensive and must share very large sets of data efficiently across many processors in parallel implementations. For example, a network with n nodes can have as many as $3^{\frac{n}{3}}$ maximal cliques [56]. The memory requirement may grow exponentially with the size of the network and may even reach terabyte scale on modest-sized genomes. Algorithms with efficient data representation that can take advantage of ultra-large shared memory architectures are apparently well suited for addressing these types of data-intensive problems in systems biology.

This dissertation provides scalable, exact, and efficient graph algorithms for genome-scale biological applications. It analyzes and integrates biological networks using graph theoretical approaches. The computational aspect concentrates on algorithm design and implementation for clique enumeration on general and bipartite graphs. The biological application aspect applies these graph algorithms to the linkage disequilibrium network analyses for the comparison of population structure and to the phenotypic ontology discovery for data integration and hypothesis discovery across species and experimental systems.

1.1 Motivation

Many biological problems can be formulated as graph theory problems, where biological objects, such as genes, proteins, or phenotypes, are represented as vertices, and biological relations, such as protein-protein interactions, are represented as edges. Examples include matching molecular substructures by *Subgraph Isomorphism* [7], finding common patterns among 3D protein structures by *Maximum* or *Maximal Common Subgraph* [63], correcting the SNP conflict graph in haplotyping by *Independent Set* [35], computing perfect phylogeny haplotypes by *Vertex Cover* [9], identifying sets of putatively pairwise co-regulated genes by *Clique* [10], and discovering biclusters in gene expression data by *Biclique* [67].

Figure 1.1 maps biological applications to graph theory problems. It also shows how these graph theory problems relate to one another. The clique problem is the key among those graph theory problems because many graph theory problems, such as subgraph isomorphism and maximum common subgraph, can be reduced to the clique problem in polynomial time. Moreover, the independent set and the vertex cover problems are closely related to the clique problem, where the former is complementary to it and the latter is a complementary dual of it.

The clique-centric model has been applied to a variety of problems in genomics and computational biochemistry, such as the recognition of 3D substructure motif in proteins [42], the identification of maximal complementary sets of hydrogen bond donor/acceptor pairs in two proteins [31], the integration of genome mapping data [37], the visualization of plant metabolomic correlation networks [46], the extraction of co-regulated genes in regulatory networks based on massive microarray data sets [1], the discovery of *cis* regulatory motif [11], and the prediction of interactions in protein networks [80]. All of these applications involve the enumeration of maximal cliques in the constructed graphs. As a result, the enumeration version of the clique problem is a focus of this dissertation.

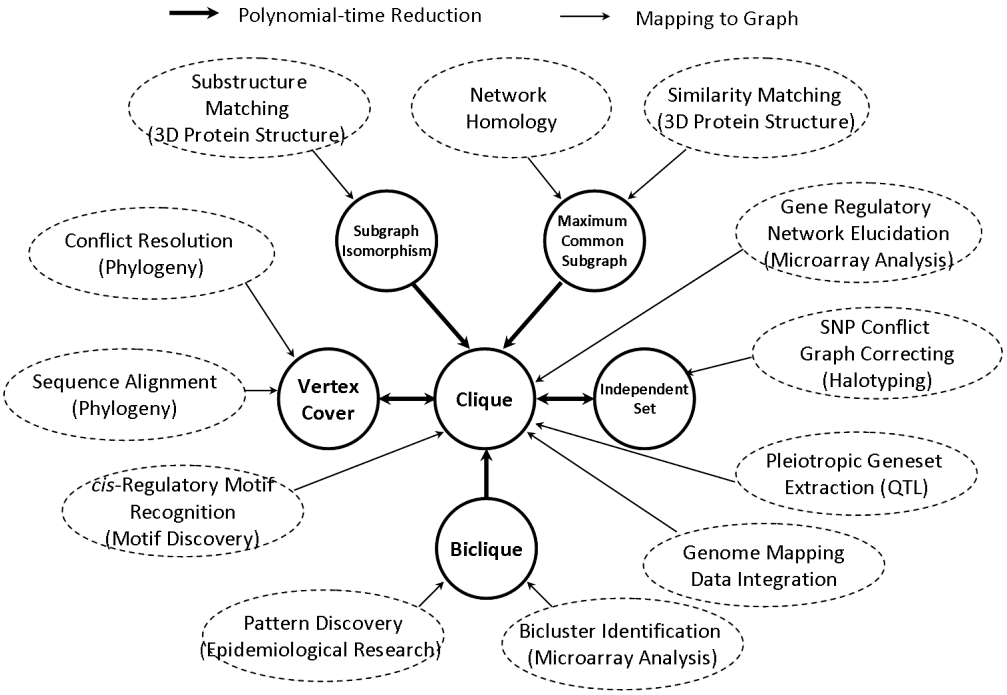


Figure 1.1: A mapping of biological applications to graph theory problems

The biclique problem is a special version of the clique problem on bipartite graphs. The integration of multiple genome-scale data sets has recently become an algorithmic challenge for modern systems biology [65, 19, 43, 57]. In such settings, bipartite graphs are often useful in representing relationships across pairs of heterogeneous data types, with the interpretation of such relationships accomplished through an enumeration of maximal bicliques. However, previously-known algorithms are highly inefficient and do not scale for this important task. The demand for efficient bipartite graph algorithms is pressing. With the biclique enumeration algorithm developed in this dissertation, we were able to begin an empirical approach to the discovery of the ontology of biological processes based on the decomposition of gene-phenotype association graphs.

1.2 Problem Definitions

We will first define the problems to be solved in this dissertation, including the clique problem and the biclique problem. The focus is on the enumeration version of both problems. Algorithms designed for them must have the following three properties:

- **Exact**: The algorithms should provide exact solutions to these computationally intensive problems instead of providing approximation ones because of the underlying value of biological data sets;
- **Scalable**: the algorithms should scale to genome-wide data sets, and parallel implementations of these algorithms should scale to multiple processors;
- **Efficient**: the algorithms need to be efficient to problems on large-scale data sets to support spontaneous scientific research ideas. Graphs generated from biological data sets tend to be large but sparse; the algorithms should deliver solutions in practical time on such graphs.

Graphs studied in this dissertation are finite, simple and undirected. For a given graph $G = (V, E)$, $V(G)$ and $E(G)$, denote the set of vertices and the set of edges of G respectively. For simplicity, V and E are often used. The *order* (or *size*) of G is the cardinality of V, denoted by n. Edges of a simple graph are uniquely identified by their endpoints. We use (u, v) to denote an *edge* whose *endpoints* are u and v. Loops that have equal endpoints (u, u) are ignored, that is, $\forall u \in V, (u, u) \notin E$. A *complete graph* is a simple graph whose vertices are pairwise adjacent; the complete graph with n vertices is denoted as K_n. A subgraph $G[C]$ *induced* by a vertex set C consists of C and all edges whose endpoints are contained in C. The *neighborhood* of a vertex v is defined as the set of its immediately connected neighbors, $N(v) = \{u \mid u \in V,\ (u, v) \in E\}$. The *degree* of a vertex v, denoted by $d(v) = |N(v)|$, is the number of vertices in its neighborhood $N(v)$. The *common neighborhood* of a subset $S \in V$ is the set of vertices that are immediately connected neighbors of all vertices in S, $N(S) = \{u \mid u \in V \setminus S,\ \forall v \in S,\ (u, v) \in E\}$. In other words, the common neighborhood of a set of vertices is the intersection of their neighborhoods, $N(S) = \bigcap_{v \in S} N(v)$. These notations are used throughout the dissertation.

1.2.1 The Clique Problem

The clique problem is the central problem to be addressed here. A *clique* in a graph G is defined as a set of vertices in G that are completely connected. That is, for every pair of vertices in the clique, there exists an edge between them. The *size* of a clique is the number of vertices it contains. Two versions of the clique problem are frequently used for the analysis of biological data sets: the *maximum clique* problem is the optimization version that requires the algorithm to

identify the largest clique in a graph; the *maximal clique* problem is the enumeration version that requires to find all maximal cliques in a graph where a maximal clique is defined as a clique that cannot be enlarged by the addition of any vertex. It should be noted that a maximum clique is one of the largest maximal cliques in a graph. The optimization problem is a well-known $\mathcal{NP}$-complete problem [32]. The maximal clique problem is equally difficult since finding all maximal cliques identifies simultaneously the cliques of maximum size. An example clique, maximal clique, and maximum clique are shown in Figure 1.2. The enumeration problem of maximal clique is formulated as follows:

MAXIMAL CLIQUE
Input : A graph $G = (V, E)$.
Output: All maximal cliques, or vertex subsets $V' \subseteq V$, such that the induced subgraph $G[V']$ is complete and there is no subset $V'' \supsetneq V'$ such that $G[V'']$ is also complete.

By definition, there is no constraint on the size of a maximal clique. Considering that cliques of interest are typically large in many biological applications, there is a potential exponential number of small maximal cliques that are of no interest in a graph. Thus, algorithms that are able to limit maximal clique enumeration by size are highly desirable. Users often choose to track the progress of algorithm execution in order to deal with computationally complex problems. One way to implement such a useful feature is to enumerate cliques in order of size.

The MCLEARS algorithm described in Section 3.2 uses the breadth-first tree search to identify all maximal cliques (**exact**) in non-decreasing order. The algorithm finds maximal cliques within a specific size range by avoiding the generation of small size cliques using the k-clique enumeration algorithm, by pruning out non-maximal candidates from the search tree, and by implementing a novel method for checking maximality to avoid pairwise clique comparison (**efficient**). To solve problems with thousands of vertices (**scalable**), the algorithm uses a compact data structure to maintain the list of cliques and a bitmap data representation to hold the neighborhoods of cliques. A parallel MCLEARS algorithm (Section 3.3) is developed on shared memory machines to lower the memory requirement of a single processor. The parallel MCLEARS algorithm achieves linear speedups (**scalable**) by taking advantage of process synchronization and load balancing techniques.

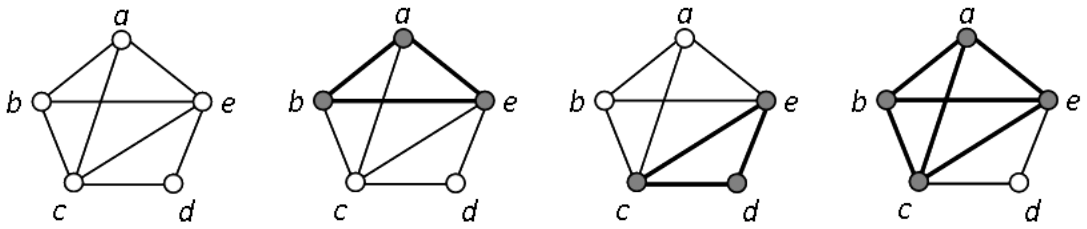
1.2.2 Problems on Bipartite Graphs

In the integration of multiple genome-scale data sets, bipartite graphs are useful in representing relationships across pairs of heterogeneous data types, and the interpretation of such relationships is accomplished through an enumeration of maximal bicliques. A bipartite graph is one whose vertices can be partitioned into a pair of non-empty, disjoint subsets such that no two vertices within the same subset are connected by an edge. A biclique is a complete bipartite graph, that is, a bipartite graph containing all permissible edges. The formal definitions are as follows:

Definition 1 A graph $G = (U \cup V, E)$ is a *bipartite graph* if $U \cap V = \emptyset$, $\forall\, u_1, u_2 \in U$, $(u_1, u_2) \notin E$, and $\forall\, v_1, v_2 \in V$, $(v_1, v_2) \notin E$.

Definition 2 Let $G = (U \cup V, E)$ be a bipartite graph, a *biclique* $C = (U', V')$ is a subgraph of G induced by a pair of two disjoint subsets $U' \subseteq U$, $V' \subseteq V$, such that $\forall\, u \in U'$, $v \in V'$, $(u, v) \in E$.

Unlike the well-known maximum clique problem, there are two distinct variants of the maximum biclique problem: the *vertex maximum biclique* problem and the *edge maximum biclique* problem. For bipartite graphs, the former is to find a biclique with the largest total number of vertices from



(a) G (b) C_1 (c) C_2 (d) C_3

Figure 1.2: (a) A graph G of order 5; (b) A clique $C_1 = \{a, b, e\}$ of size 3 in G, which is not maximal because it is a subgraph of C_3; (c) A maximal clique $C_2 = \{c, d, e\}$ of size 3 in G; and (d) A maximum/maximal clique $C_3 = \{a, b, c, e\}$ of size 4 in G. Graph G has a total number of two maximal cliques, C_2 and C_3.

each vertex set and the problem can be solved in polynomial time [32], while the latter is to find the one with the largest number of edges and it is $\mathcal{NP}$-complete [62].

A *maximal biclique* in a bipartite graph is a complete bipartite subgraph that is not contained in any other complete bipartite subgraph. The enumeration version of the biclique problem is to find all maximal bicliques in a bipartite graph. An example of maximal and maximum bicliques is shown in Figure 1.3. All edge maximum bicliques and all vertex maximum bicliques can be generated by solving the enumeration problem. Thus, one of the most important elements of this study is the enumeration version of the maximal biclique problem, which is formalized as follows:

> MAXIMAL BICLIQUE
> **Input :** A bipartite graph $G = (U \cup V, E)$.
> **Output:** All maximal bicliques, or subsets U' of U and V' of V, for which the induced subgraph $G[U' \cup V']$ is complete, and there are no subsets $U'' \supseteq U'$, $V'' \supsetneq V'$ or $U'' \supsetneq U'$, $V'' \supseteq V'$ such that $G[U'' \cup V'']$ is also complete.

As observed in [24], the maximal biclique enumeration problem cannot be solved in polynomial time simply because the number of maximal bicliques can be exponential in the input size. The solution requires efficient algorithms to provide exact solutions to large-scale datasets in practical time. The MBEA algorithm described in Chapter 4 finds all maximal bicliques (**exact**) by exploiting the inherent structure in bipartite graphs. The algorithm uses a branch-and-bound technique to prune non-maximal candidates on the search tree, which is further pruned by directly removing dominated vertices from the candidate set and by selecting candidates in the order of their degree (**efficient**). The reduced search space enables MBEA to scale to large biological data sets. Experimental results show that the average delay time for outputting a maximal biclique by MBEA is almost constant on graphs generated from biological data with tens of thousands nodes (**scalable**).

1.3 Applications

The clique and biclique enumeration algorithms are applied to two applications in genetics: the linkage disequilibrium (LD) analysis in population genetics with clique algorithms, and the ontology discovery for systems genetics with biclique algorithms. Both applications require exact solutions. The study of LD networks analyzes large sets of genetic variations throughout the genome, and the

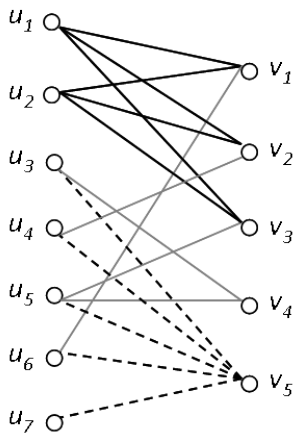


Figure 1.3: A bipartite graph G has an edge maximum biclique $B_1 = (\{u_1, u_2\}, \{v_1, v_2, v_3\})$ with 5 vertices and 6 edges, and a vertex maximum biclique $B_2 = (\{u_3, u_4, u_5, u_6, u_7\}, \{v_5\})$ with 6 vertices and 5 edges. Both B_1 and B_2 are maximal bicliques.

study of ontology integrates multiple genome-scale data sets of heterogeneous types such as gene expression data and phenotype data. Thus the algorithms must be exact and scalable.

In both applications a matrix of association scores is calculated by statistical metrics (e.g. linkages, correlations) for pairs of biological objects (e.g. genes, phenotypes). The matrix is modeled as an unweighted graph after being transformed into a binary matrix by a threshold. Clique and biclique algorithms are then employed to identify networks of highly related objects in the graph. The resulting networks are further analyzed in the context of their applications. The roles of clique and biclique algorithms in these biological applications are shown in Figure 1.4.

In LD analysis, two genetic loci are said to be associated in LD when some of their allele combinations occur more often than expected assuming the loci segregate completely independently. The identification of LD networks, which contain groups of associated loci, is crucial to characterizing the biological drivers that contribute to LD. Using clique algorithms we comprehensively detect high-dimensional long-range LD networks in the graphs constructed from large sets of genetic markers. Linkages are represented as an edge-weighted graph, with vertices representing genetic loci and edge weights representing linkages. This method allows the extent of LD at varying edge-weight thresholds to be evaluated. The size, distribution, and complexity of the detected LD networks can be analyzed to compare populations and sub-populations quantitatively.

The study of ontological discovery seeks to identify relations among genes and their roles in the biological processes. In this setting, gene-sets from various sources are regarded as phenotypes. The ontology of these gene-sets can be achieved by the creation and decomposition of gene-phenotype association graphs derived from a matrix of scores for the associations of genes to phenotypes, such as correlations, p-values, q-values, or binary values. Binary matrices derived by applying thresholds to the scores are represented as bipartite graphs. Using the MBEA algorithm, networks of phenotypes and the common genes with which they are associated are computationally derived and integrated into a directed acyclic graph of phenotype relationships. This provides an empirical discovery of the natural phenotype ontology.

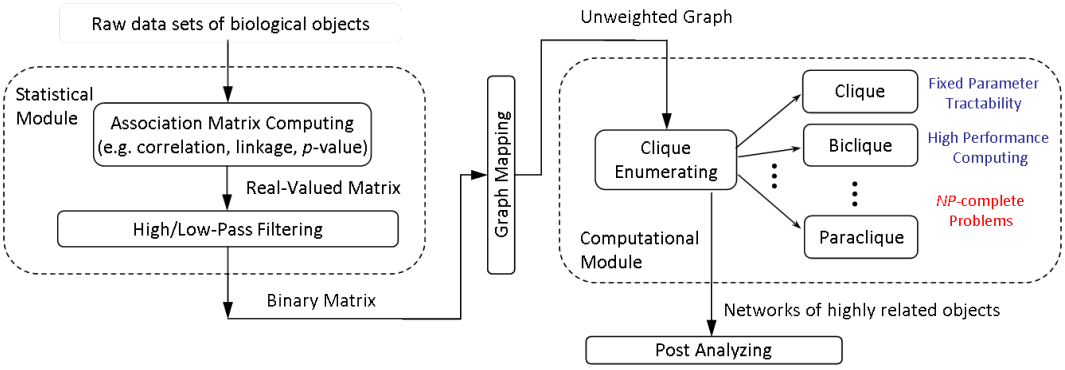


Figure 1.4: Clique and biclique enumeration algorithms in biological applications

1.4 Contributions

The goal of this study is to provide exact, scalable and efficient solutions for $\mathcal{NP}$-hard graph problems in large-scale biological applications. Algorithms are designed for the clique and biclique enumeration problems and are applied in two important applications in genetics: the linkage disequilibrium network analysis and the ontology discovery for heterogeneous data integration. The contributions of this dissertation can be summarized as follows:

An algorithm for clique enumeration A scalable and exact algorithm, MCLEARS [83] is developed to enumerate maximal cliques of ranged sizes in non-decreasing order so that the generation of exponential number of undesired maximal cliques can be avoided. A sequential implementation of MCLEARS is about 300 times faster than the previous algorithm that provided the same feature [46]. The parallel implementation on ultra-large globally addressable memory architectures with dynamic load balancing techniques provides a linear speedup.

An algorithm for biclique enumeration MBEA is the first general-purpose exact algorithm designed to enumerate all maximal bicliques in a bipartite graph [84]. Unlike other approaches, the algorithm neither places undue restrictions on the problem nor inflates the problem size. Efficiency is achieved by exploiting the inherent structure of bipartite graphs, and by ensuring that biclique enumeration avoids duplication while pruning non-maximal candidates. Experiments using both biological data and synthesis data indicate that MBEA can be as much as three orders of magnitude faster than the best alternatives [6, 86].

Linkage disequilibrium networks analysis using clique algorithms The clique-centric model is used in the linkage disequilibrium analysis to compare the structure of inbred mouse populations [85]. Using MCLEARS, long-range LD networks are comprehensively enumerated throughout the genome from large-scale sets of genetic variations and are quantitatively analyzed to compare population structure. This goes beyond the traditional LD studies that focus on pairs or small blocks of loci.

Phenotype ontology discovery using biclique algorithms The MBEA algorithm has been employed in the "Ontological Discovery Environment" (`http://ontologicaldiscovery.org`), a web-based tool that allows public users in the biological sciences to upload sets of genes associated with phenotypic descriptors for the derivation of their own hierarchical trees of the phenome space.

Maximal sets of phenotypes, with the genes with which they are associated, are computationally derived by the MBEA algorithm and integrated into a directed acyclic graph that represents the phenotype ontology. Without a fast implementation of the biclique enumeration algorithm, such a tool would not be possible.

1.5 Document Organization

The remainder of this dissertation presents the details of this research. Chapter 2 reviews clique algorithms on general and bipartite graphs followed by a survey of their applications in computational biology. Chapter 3 presents MCLEARS, an algorithm for clique enumeration and its parallel implementation. Chapter 4 describes MBEA, the first general-purpose algorithm for biclique finding in bipartite graphs. Chapter 5 provides the clique-based model for the linkage disequilibrium analysis with MCLEARS and the biclique-based model for phenotypic ontological discovery with MBEA. Chapter 6 summarizes works done in this study and discusses possible directions for further research.

Chapter 2

Literature Review

The study of clique algorithms is a classic area in graph theory research. Both optimization and enumeration problems have important applications in many domains. This chapter reviews the exact algorithms for clique and biclique enumeration. Biological applications of these problems are reviewed as well.

2.1 Algorithms for Clique Problems

The maximum clique problem is one of the earliest identified $\mathcal{NP}$-complete [32] problems. An elaborative survey on algorithms, complexity, and applications of this problem can be found in [12]. The enumeration problem of finding all maximal cliques is at least as difficult as finding the largest clique. It has been demonstrated [56] that a graph can contain an exponential number (as many as $3^{\frac{n}{3}}$) of maximal cliques. Also, [40] showed that a polynomial-delay time algorithm for enumerating maximal cliques in reverse lexicographic order does not exist if $\mathcal{P} \neq \mathcal{NP}$.

Surveys on maximal cliques can be found in [61] and [12]. These studies summarized the development of exact enumerative algorithms for maximal cliques and maximal independent sets (MISs) in the 1970s and 1980s. Enumerating maximal cliques (also called clique-finding) in a graph is equivalent to finding MISs in its complement graph. In the 1970s several maximal clique algorithms were compared experimentally and theoretically [41]. Recently, some of the popular clique-finding and MIS-finding algorithms were compared experimentally [38] on graphs generated from biological data sets.

2.1.1 A Taxonomy of Maximal Clique Algorithms

Most of the clique-finding algorithms can be classified into three categories based on the method they use: (a) the *point removal method*, (b) the *backtracking method*, or (c) a combination of both. Other methods include graph decomposition, breadth first tree search, and matrix multiplication. These clique-finding and MIS-finding algorithms are listed in Table 2.1.

The Point Removal Method

The *point removal method*, also called the *vertex sequence method*, produces cliques of a graph G from the cliques of $G\backslash\{v\},\ v \in V$. This method tends to find cliques of smaller size earlier. This method requires large memory to maintain the list of cliques found in core because cliques must be compared with those previously produced to ensure the maximality (the property of a clique that it is maximal or not). Algorithms based on this method include those in [13], [8], and [59].

Table 2.1: Algorithms for enumeration of maximal cliques or maximal independent sets

Algorithms	Clique or MIS	Methods	Features
Bierston [8]	Clique	Point removal	Requires to maintain a list of cliques in core; smaller cliques tend to be found earlier
Osteen-Tou [59, 58]	Clique	Point removal	Improved Bierstone algorithm
Akkoyunlu [5]	Clique	Backtracking	Operates on symbolic expressions; hard to implement
Bron-Kerbosch (BK) [14]	Clique	Backtracking	Larger cliques tend to be found earlier; polynomial storage
Johnston [41]	Clique	Backtracking	Simplified & generalized BK; experimental comparison of BK and its variants; theoretical comparison with other algorithms.
Gerhards-Lindenberg [33]	Clique	Backtracking	A variant of BK that generates cliques from partial cliques related to fixed vertices
Koch [44]	Clique	Backtracking	Applied a variant of BK on the maximal common subgraph problem
Harley et al. [37, 38]	Clique	Backtracking	A variant of BK with adjacency list; more efficient on genome overlap graphs
Tomita et al. [71]	Clique	Backtracking	A variant of BK with worst-case time $O(3^{n/3})$
Wan et al. [76]	Clique	Depth first search	Utilizes idea of cluster coefficients; designed for complex network
Chiba-Nishizeki [21]	Clique	Point removal with backtracking	An improvement of Tsukiyama et al. with $O(a(G)m\mu)$ time, where $a(G)$ is the arboricity of G
Meeusen-Curyvers [55]	Clique	Graph decomposition	Designed for directed graph
Kose et al. [46]	Clique	Breadth first search	Requires to maintain a list of previously found cliques
Makino-Uno [54]	Clique	Matrix multiplication	$O(M(n))$ delay time and $O(n^2)$ storage, where $M(n)$ is time to multiply two $n \times n$ matrices.
Loukakis-Tsouros [53, 52]	MIS	Backtracking	Finds cliques in lexicographical order
Johnson et al. [40]	MIS	Depth first search	$O(n^3)$ delay time
Eppstein [25, 26]	MIS	Depth first search	Finds MISs of size smaller than a given parameter
Tsukiyama et al. [72]	MIS	Point removal with backtracking	$O(mn\mu)$ time, $O(n+m)$ storage
Lawler et al. [47]	MIS	Point removal with backtracking	Generalized the results in Tsukiyama et al.

The Backtracking Method

The *backtracking method* is a more popular method employed by many clique-finding algorithms. It systematically explores a search tree of possible solutions by means of a depth first search (DFS). This method not only eliminates the generation of duplicated cliques, but also avoids the exploration of some branches that cannot lead to new solutions. In addition, the backtracking method requires only polynomial storage space.

In 1973, two algorithms utilizing the recursive backtracking strategy were proposed [5, 14]. The algorithm presented in [5] operates on symbolic expressions and thus is difficult to implement. The algorithm described in [14](BK) has a storage requirement of at most $O(\frac{1}{2}n(n+3))$, and an improved BK algorithm tends to find larger cliques first. Many algorithms [41, 33, 70, 44, 37] proposed after that are variants of the algorithms in [14]. Other efforts include a backtracking algorithm for generating all MISs in lexicographical order [53], a DFS-based algorithm for finding MISs of a size smaller than a given parameter [25], and most recently, a DFS-based maximal clique algorithm for complex networks [76].

Algorithms in this category share the same advantage with DFS in that the storage requirement is polynomial, but they have the same drawback in that cliques of different sizes are discovered in a quasi-random fashion.

A Combined Method

An algorithm that combines the point removal method and the backtracking method was first proposed in 1977 [72] to find all MISs in a graph. This algorithm has a theoretical time complexity of $O(mn\mu)$ and a storage requirement of $O(n+m)$, where n, m and μ are the numbers of vertices, edges and MISs, respectively. Subsequently, a more efficient algorithm based on the algorithm in [72] was proposed in [21] for generating all maximal cliques of a graph G in $O(a(G)m\mu)$ time, where $a(G)$ is the arboricity (the minimum number of forests needed to cover all the edges of the graph) of G and μ is the number of maximal cliques in G.

Other Methods

Other algorithms for finding maximal cliques include graph decomposition [55], matrix multiplication [54], and breadth first tree search [46]. The algorithm in [46] enumerates maximal cliques in non-decreasing order of size, which is a feature not available in other methods. However, this method shares the same drawback with the point removal method in that all cliques must be maintained in memory for checking maximality. The MCLEARS algorithm described in Chapter 3 is also based on the breadth first search. It finds maximal cliques in non-decreasing order of their sizes but with a lower storage requirement than the one in [46].

2.1.2 Efficiency Comparison

It has been shown [41, 38, 1] that the backtracking-based BK algorithm [14] and its variants are the most efficient. The BK algorithms have been generalized in [41], and a family of clique-finding algorithms based on them were described in the study. The research established that the so-called *simplified BK* algorithm is the most efficient for large sparse graphs whose edge density is less than 25%. The study also analyzed theoretically four other enumeration algorithms, including those in [5] and [59], and concluded that none were more effective than the BK algorithm.

In [38], three BK algorithms including the improved BK algorithm [14], the *simplified BK* [41], and a modified BK algorithm by using adjacency list to represent the graph [37], were compared

with three other algorithms [72, 53, 21], two of which were based on a combination of the point removal and the backtracking methods. The experiment results revealed that (a) the modified BK algorithm is the most efficient on large, sparse graphs that arise in genomic mapping data; and (b) the family of BK algorithms is faster than the other three algorithms. In [1], the relative efficiency of the BK algorithms and the algorithm in [46] were evaluated on microarray gene expression data. Results confirmed that the BK algorithms are more efficient.

According to the results of the survey above, the backtracking method-based algorithms, especially the BK algorithms, are the most efficient in finding maximal cliques of a graph. However, these algorithms discover maximal cliques in a quasi-random fashion. In biological applications with large data sets, algorithms seeking to discover maximal cliques in the order of their sizes are preferred. Although the MCLEARS algorithm described in Section 3.2 is not as efficient as the BK algorithms, it can find maximal cliques in non-decreasing order. This is important when dealing with computationally complex problems because the user can track the execution progress of the algorithm.

2.1.3 Parallel Algorithms

The clique-finding problem has been well studied. However, only a few parallel algorithms can be found in the literature. One parallel algorithm was proposed in [15] to generate maximal cliques of a circle graph. The algorithm uses a coarse-grained multicomputer model and requires $O(\log p)$ communication rounds where p is the number of processors. This algorithm takes advantage of the special structure of circle graphs, which is not applicable to general graphs and not practical for graphs from biological applications. In addition, the study offered no experimental results. Another parallel algorithm was investigated in [23] for finding maximal cliques in large complex networks based on the maximal clique algorithm in [76]. However, the parallel model of the algorithm simply distributes the subtrees rooted at every vertex of the search tree to multiple processors. There is no load balancing strategy used in the algorithm to ensure balanced workloads. The experimental results showed that the parallel implementation does not scale well when using 30+ processors.

There have been some efforts to parallelize the maximum clique problem. The parallel exact algorithm presented in [60] solves the maximum clique problem using the Message Passing Interface (MPI). The algorithm uses a master/worker programming model: The master distributes preprocessed sub-tasks to workers and no communication is required between workers. The experimental results showed that the speedups ranged between 1.82 and 2.66 for unweighted graphs. However, the algorithm was tested using no more than 4 processors, which was not enough to determine whether the program could scale to a larger number of processors. A parallel algorithm using dynamic load balancing techniques was proposed in [3, 4] for the minimum vertex cover problem. The best performance results were obtained by directly launching secure shells on 32 processors at most.

The parallel MCLEARS algorithm described in Section 3.3 uses load balancing techniques in a master/worker framework to ensure balanced work across multiple processors. The algorithm is implemented with threads on shared memory machines to take advantage of the large, globally accessible memory architecture. The experimental results show the parallel implementation scales well up to 128 processors.

2.2 Algorithms for Biclique Problems

A variety of computational challenges in systems biology can be addressed by finding maximal bicliques in bipartite graphs. Applications of this approach include microarray data biclustering

[17, 67, 77], maximal concatenated phylogenetic datasets determination [65], gene-phenotype associations elucidation [19], proteome-transcriptome relationships analysis [43], and pattern discovery in epidemiological research [57].

However, despite these kinds of applications no general-purpose biclique enumeration algorithm aiming at bipartite graphs has been described in the literature. Existing algorithms for solving this problem fall into two major categories: (a) algorithms designed for bipartite graphs but either placing restrictions on the problem or requiring the reduction of the biclique problem to other problems, and (b) those designed for general graphs and thus unable to take advantage of the inherent structure of bipartite graphs. Table 2.2 lists these maximal biclique algorithms, the inputs and outputs (with restrictions if there is any), and the methods that they use.

2.2.1 Algorithms for Bipartite Graphs

Algorithms for finding maximal bicliques on bipartite graphs usually take one of the following approaches: (a) exhaustive search with restrictions on outputs, (b) reduction to the clique enumeration problem on general graphs, and (c) reduction to the frequent itemset mining problem in transaction databases.

Exhaustive Search with Problem Restrictions

The most intuitive method is to thoroughly build all subsets of one vertex set, finding their intersections in the other vertex set and examining the maximality. Algorithms based on an exhaustive search have to place some restrictions on the problem to reduce the enormous search space. Moreover, that kind of comprehensive search requires storing generated bicliques in order to determine their maximality.

An iterative algorithm was suggested in [65] for building subsets from pairs of vertices to subsets of larger sizes. The algorithm limited the sizes of both vertex sets in bicliques to reduce the search space, but still required sizeable memory to maintain the list of found bicliques for generating larger bicliques and for examining the maximality of bicliques.

The algorithm described in [57] built bicliques based on set expansion and extension operations. It employed a hash table to determine the maximality that avoids pairwise biclique comparison, and a queue to maintain bicliques prioritized by figure-of-merit (FOM) values (e.g. p-values). Users could specify constraints on the FOM values in order to filter bicliques of no interest.

Reduction to Clique Enumeration on General Graphs

The second approach relies on graph inflation. As observed in [54], the enumeration of maximal bicliques on a bipartite graph can be turned into the enumeration of maximal cliques on a general graph by adding edges to the input until each of the two disjoint vertex sets is transformed into a clique. However, this approach is neither practical nor scalable due to the enormous number of edges that may be needed and the concomitant increase in problem difficulty that is incurred.

Given a bipartite graph $G = (U \cup V, E)$ where $|U| = m,\ |V| = n,\ |E| = e$, the number of edges needed to transform G to a corresponding graph $\hat{G}$ is $\binom{n}{2} + \binom{m}{2}$. Thus this method transforms the problem of finding maximal bicliques in a bipartite graph with edge density $d(G) = \dfrac{e}{m \times n}$ to the problem of finding maximal cliques in a graph $\hat{G}$ with edge density $d(\hat{G}) = \dfrac{e + \binom{n}{2} + \binom{m}{2}}{\binom{m+n}{2}}$. It should be noted that $\hat{G}$ might be dense even if G is sparse. When G has two vertex sets of equal

Table 2.2: Algorithms for enumeration of maximal bicliques

Algorithms	Inputs	Outputs	Methods
Sanderson et al. [65]	Bipartite graph	Maximal bicliques of bounded minimum size	Exhaustive search by iterative bicliques building
Mushlin et al. [57]	Bipartite graph	Maximal bicliques of bounded sizes and FOM	Exhaustive search with a priority queue
Zaki et al. [82]	Bipartite graph	Maximal bicliques	Frequent closed itemset mining in transaction databases
Makino & Uno [54]	Bipartite graph	Maximal bicliques	Maximal clique finding in general graphs
Eppstein [24]	General graph of bounded arboricity	Maximal bicliques	Exhaustive search
MICA [6]	General graph	Maximal bicliques	A consensus algorithm
MineLMBC [50]	General graph	Maximal bicliques of bounded minimum size	A divide-and-conquer approach

size and no edges (i.e. $|U| = |V| = n,\ |E| = 0$), $\hat{G}$ has a density $\frac{n^2 - n}{2n^2 - n}$, which is close to 50%. Figure 2.1 (a,b) illustrates the correspondence between these two problems.

Reduction to Frequent Itemset Mining

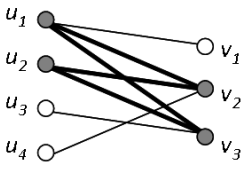
The third approach comes from the field of data mining. It was observed in [82] that a transactional database can be represented by a bipartite graph. The paper showed a one-to-one correspondence between frequent closed itemsets of a transaction database and maximal bicliques of a bipartite graph. A subset of items is defined as a *frequent itemset* if it occurs in at least one transaction. On one hand, a frequent itemset and the set of transactions containing the frequent itemset form a biclique. On the other hand, the adjacency lists of a bipartite graph can be viewed as a transaction database by treating each vertex in one set as an item and each vertex in the other set as a transaction that contains a subset of items. A biclique can thus be mapped to a frequent itemset. A maximal biclique corresponds to a frequent closed itemset, where a frequent itemset I is said to be *closed* if the set of transactions containing I do not contain a superset of I. Figure 2.1 (a,c) shows a mapping between these two problems.

It was further proven in [49] of the correspondence between maximal bicliques of a general graph and frequent closed itemsets. The study recommended frequent itemset mining (FIM) techniques [81, 78, 34, 86, 73] to enumerate maximal bicliques. However, this approach needs a post-processing step to obtain the transaction set for each frequent closed itemset because FIM algorithms produce only frequent itemsets. The post-processing step can be costly when both vertex sets of a bipartite graph are large.

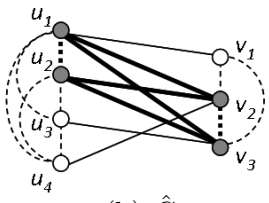
2.2.2 Algorithms for General Graphs

The other way to find maximal bicliques in bipartite graphs is to treat bipartite graphs as general graphs. Algorithms designed for general graphs are applicable to bipartite graphs but will not be efficient because the structure of bipartite graphs is not utilized.

The maximal biclique enumeration problem was studied theoretically in [24] on graphs of bounded arboricity. The research ascertained that all maximal bicliques of a graph can be enumerated in time $O(a^3 2^{2a} n)$, where a is the arboricity of the graph and n is the number of vertices in the graph. Although the algorithm has a linear complexity when the arboricity of the input graph is bounded, it is not practical for large graphs since a can easily be around 20 in practice [82].



(a) G



(b) $\hat{G}$

Transactions	Items
v_1	u_1
v_2	u_1, u_2, u_4
v_3	u_1, u_2, u_3

(c) $TD(G)$

Figure 2.1: (a) A bipartite graph G with a maximal biclique $B = (\{u_1, u_2\}, \{v_2, v_3\})$. (b) The graph $\hat{G}$ transformed from G by adding edges (dashed lines) to all pairs of vertices in the same set. Biclique B in G becomes a maximal clique in $\hat{G}$. (c) The transaction database $TD(G)$ mapped from G by treating V as the set of transactions and U as the set of items. Biclique B becomes a frequent closed itemset in $TD(G)$.

A suite of consensus algorithms were proposed in [6] for finding complete-though not necessarily induced-bipartite subgraphs of a graph. However these algorithms must keep all maximal bicliques in memory. The Modular Input Consensus Algorithm (MICA), the most efficient among them, has the space complexity $O(\mathcal{B})$ and the time complexity $O(n^3\mathcal{B})$, where $\mathcal{B}$ is the number of maximal bicliques.

An algorithm (MineLMBC) based on divide-and-conquer was proposed in [50] to mine large maximal bicliques from general graphs by putting size constraints on both vertex sets in order to iteratively prune the search space. The algorithm reduces the space complexity to $O(n^2)$ and the time complexity to $O(n^2\mathcal{B})$. The algorithm on dense graphs from 2nd DIMACS Challenge benchmarks outperforms the MICA algorithm when the minimum biclique sizes are constrained by certain thresholds.

To solve the biclique enumeration problem, restrictions on either inputs or outputs have been proposed to reduce the search space. These limits include bounding the maximum input degree [67], bounding an input's arboricity [24], bounding the minimum biclique size [50, 65], and bounding the FOM value of bicliques [57]. Of course no algorithm relying on these restrictions can solve arbitrary bipartite instances.

The MBEA algorithm described in Chapter 4 exploits the inherent structure in bipartite graphs to achieve efficiency without any restriction on inputs or outputs. Performance results for both biological and synthetic graphs show that the MBEA algorithm is as much as two to three orders of magnitude faster than MICA [6], the best algorithm for finding bicliques on general graph with no restrictions, and FPclose [34, 86], a state-of-the-art algorithm for mining closed itemsets.

2.3 Applications

The utility of clique and biclique algorithms has emerged in many application domains [12]. This study focuses on those in computational biology. This section reviews the biological applications of the maximal clique problem and the maximal biclique problem.

2.3.1 Applications of Clique Enumeration

With advances in genomics and computational biochemistry, maximal clique enumeration algorithms have been applied to a variety of graphs, including those from protein structures [31, 36, 39, 42, 45, 66, 79], from genome mapping data [37, 38], from metabolomic correlations [46], from microarray gene expression data [1, 10, 19, 20, 75], and from protein interactions [80].

Protein common structural pattern recognition. Maximal clique enumeration algorithms have been widely used since the early 1990s for 3D motif searches in proteins at the secondary structure level [36, 39, 42, 45, 66, 79]. Comparison of proteins in a structure level may reveal common geometric arrangements of the secondary structures and thus provide clues to functional relationships between proteins. A protein is modeled as a graph where the secondary structure elements (SSEs), including α helices and β strands, are mapped as vertices, and the spatial and angular relationships between SSEs are mapped as edges. A "correspondence graph" is then constructed for each pair of protein graphs by mapping every pair of nodes in the protein graphs as a vertex if the pair of SSEs are of the same type, and two vertices in the correspondence graph are connected if the differences in angles and distances between the two pairs of SSEs are below user-defined thresholds. A maximal clique detected in the correspondence graph is equivalent to a common structural pattern in the pair of proteins.

Protein docking. In [31], clique-finding algorithms were employed to solve the protein docking problem, which is to determine whether two proteins interact to form a stable complex and, if they do, to determine how. To address this problem, a docking graph is constructed for two proteins, where vertices represent a set of potential hydrogen bond donor and acceptor atoms, and a pair of vertices is connected if the difference in distance between the two atoms in the pair of proteins is within a predefined tolerance. A maximal clique in the docking graph corresponds to a maximal set of possible hydrogen bonds.

Genome mapping data integration. The integration of genome data becomes an important problem due to the differences between two types of methods for mapping genome data: overlap-based and probe-based. In [37], the overlap data were effectively converted to probe-like data by finding maximal sets of mutually overlapping clones (i.e. small DNA fragments in the process of mapping). The clone overlaps were modeled as an interval graph, where vertices corresponded to the intervals and edges corresponded to pairs of intervals that have intersections. A maximal clique in the interval graph is a maximal set of mutually overlapping clones that share a common region of the genome, called a "virtual probe."

Plant metabolomic correlation network visualization. Profiling gene products to the metabolite level may be a key to clarifying novel gene functions [68]. It was proposed in [46] to visualize plant metabolomic correlation networks by means of clique-metabolite matrices. Metabolite correlations were modeled as a graph with vertices representing metabolites and edge weights representing correlation coefficients between pairs of metabolites computed from metabolite levels of a series of physiological snapshots. A threshold was then applied to the correlations in constructing an unweighted graph. A maximal biclique in the correlation graph thus corresponds to a maximal set of tightly co-regulated metabolites. The correlation network can then be visualized using the clique-metabolite membership matrix.

DNA sequence motif discovery. The availability of complete genomic sequences opens the door for computational methods to predict transcription factor binding sites. A suite of computational tools was developed in [11] using maximal cliques for motif discovery in order to identify functional DNA in non-coding regions by sequence comparison. In this work, a graph was constructed for a specific motif length or pattern, where vertices corresponded to subsequences extracted from input DNA sequences, and edges were extracted from the set of maximal subsequences combined from overlapped and adjacent subsequences. Clusters of motifs can then be identified in the graph using maximal clique algorithms. These clusters correspond to transcription factor binding sites that are either known or novel.

Microarray gene expression data elucidation. Recent efforts have extended the applications of the maximal clique problem to the analysis of microarray gene expression data. Clique-finding algorithms were used for elucidation of genetic co-expression networks from microarray data [1, 10]. Correlations for pairs of genes were computed and represented as a graph, from which cliques were extracted to represent clusters of genes having similar changing patterns of expression levels. In this framework, the regulatory loci underlying the shared genetic mediation of gene expression can be identified by combining clique data with Quantitative Trait Loci analysis [20, 19]. Clique algorithms have also been used to extract gene networks for low-dose ionizing radiation and to identify the gene sets that are impacted by radiation exposure through the comparison of cliques found in control and exposed mouse samples [75].

Protein interactions prediction. Protein-protein interactions discovered using large-scale, high-throughput experiments typically suffer from a relatively high level of noise [74]. It was proposed in [80] that the quality of these datasets could be improved by predicting protein-protein interactions using so-called "defective" cliques (nearly complete subgraphs). In this setting, proteins are mapped as vertices and interactions detected by experiments between a pair of proteins are mapped as edges. A maximal clique corresponds to a maximal set of proteins that have pairwise interactions; a defective clique is generated by merging two overlapped maximal cliques with constraints on the clique size and the number of missing edges. Missing edges in a defective clique are predicted as interactions.

2.3.2 Applications of Biclique Enumeration

Algorithms for the maximal biclique problem have been applied on bipartite graphs constructed from microarray gene expression data [17, 51, 67, 77], from sequence databases [65], from proteome-transcriptome relationships [18, 43], and from factor-individual associations in clinical surveys [57].

Microarray data biclustering. In gene expression data, the concept of bicluster was introduced as a subset of genes and a subset of conditions with a high similarity score [17]. To detect biclusters in large expression datasets, it was proposed [51, 67, 77] that graph theoretic approaches could identify bicliques in the bipartite graph representation of expression data by modeling genes and conditions as vertices on two sides and expression levels as edge weights.

Maximal concatenated phylogenetic datasets discovery. A phylogenetic tree shows the evolutionary interrelationships among various specious or other entities that are believed to have a common ancestor. The accuracy of phylogenetic tree reconstruction can be improved through the concatenation of the maximal multi-gene data sets obtained from a collection of sequences. Maximal data sets that contain at least k genes sampled from at least m species are extracted from sequence databases. This is equivalent to finding maximal bicliques of vertex set sizes no less than k and m. A method has been developed to discover complete phylogenetic data sets by enumerating all the maximal bicliques satisfying the size constraints [65].

Proteome-transcriptome relationships identification. Bipartite graphs and bicliques have been used to extract the relations between gene expressions and proteins [43]. The goal is to locate mRNA abundance that varies with disease symptoms and to identify the proteins that may be produced by individuals with particular mRNA up-regulation. Correlations between microarray expressions with peak intensities form a bipartite graph; maximal biclique algorithms were applied to identify relationships among the two data classes represented in the bipartite graph. These bicliques represent the largest sets of completely connected mRNA and protein peaks and allow forming biological meaningful hypothesis for the connection between mRNA and protein expressions.

Pattern discovery in epidemiological research. In epidemiological research, the maximal biclique enumeration for pattern discovery is especially useful when applied to case-control studies involving categorical features such as genotypes and exposures. A biclique-based approach was recommended in [57] for finding associations among multiple factors and groups of individuals (cases and controls) in a clinical survey. Bipartite graphs were used to capture the relationships between a set of individuals and the values of multiple factors including inherited genotypes, somatic

genotypes, demographic characteristics, and exposures. Maximal bicliques in such graphs represent the largest sets of people sharing a common set of features.

Chapter 3

Scalable Enumeration Algorithms

The $\mathcal{NP}$-hard nature of the problems studied in this dissertation demands that the exact algorithms used are computation intensive. In addition to being computationally complex, many of the enumeration algorithms are particularly memory intensive. A graph with n nodes can have as many as $3^{\frac{n}{3}}$ maximal cliques [56], thus the memory requirement may grow exponentially with the size of the graph and may reach terabyte scale on even modest-sized genome scale graphs. There is a clear need for at least one terabyte of memory during the analysis of the Affymetrix U74Av2 microarray data to test 12,422 probe sets in samples from the brain of *Mus musculus* [83]. In order to deal with such large memory requirements an out-of-core algorithm [1] was developed based on the recursive branching procedure suggested by [46] in their analysis of metabolomic correlation networks. However, the algorithm could not finish after one week of execution on the graphs derived from the microarray data using various threshold values for correlation of genes. The large data structure used in the algorithm cannot fit into the memory, thus intensive disk I/O access has become the major bottleneck.

To address these computation and memory obstacles to genome-scale elucidation of biological networks, *Clique Enumerator* in this chapter proposes exact and scalable solutions to the problem of maximal clique enumeration by taking advantage of: (a) the theory of fixed parameter tractability; (b) a novel bitmap data representation scheme for memory saving and search space reduction; and (c) ultra-large globally addressable memory architectures such as SGI Altix. Algorithms developed under *Clique Enumerator* for maximal clique enumeration include an algorithm to enumerate both maximal and non-maximal cliques of a given size (k-clique enumeration algorithm, see Section 3.1.2), an algorithm to enumerate maximal cliques of sizes in a given range with a novel bitmap representation scheme (MCLEARS, see Section 3.2), and a multithreaded parallel algorithm that takes advantage of large globally addressable memory architectures (parallel MCLEARS, see Section 3.3).

3.1 The *Clique Enumerator*

By definition, there is no constraint on the size of a maximal clique. Considering that the cliques of interest are typically large [11] and that there are an exponential possible number of small maximal cliques in a graph, it is reasonable to assume that a method to limit the minimum size of maximal clique is desirable. Furthermore, it is preferable to discover maximal cliques in the order of size, which is an essential feature when dealing with computationally complex problems because users can track the progress of the algorithm execution and resume the program from any clique size if the program is halted.

The Bron-Kerbosch (BK) algorithms [14] are among the most efficient algorithms when compared to other popular maximal clique enumeration algorithms [37, 1]. Both BK algorithms, subsequently referred to as *Base BK* and *Improved BK*, discover maximal cliques in a quasi-random fashion, despite the tendency of *Improved BK* to discover larger cliques first. Therefore, neither algorithm satisfies the requirement for an algorithm to discover maximal cliques in the order of size.

Clique Enumerator is proposed to address the two issues above with three steps. It avoids the generation of maximal cliques of undesired sizes and allows users to track progress of the algorithm.

1. A reasonable upper bound on clique size is determined with efficient maximum clique algorithms. Algorithms using the theory of fixed parameter tractability are discussed in Section 3.1.1.

2. All maximal and non-maximal cliques of a given size k are enumerated, where k is the user-supplied lower bound. Non-maximal k-cliques are generated for the purpose of further enumeration of maximal cliques of larger sizes. An algorithm modified from the BK algorithms is described in Section 3.1.2.

3. Using the non-maximal k-cliques as input, maximal cliques are enumerated in increasing order of size until the upper bound is reached. An algorithm using a novel bitmap data representation scheme is discussed in Section 3.2.

The algorithm in Step 3 can be parallelized to achieve performance speedups with multiple processors. A parallel algorithm with multithreaded implementation is described in Section 3.3. It takes advantage of large, globally addressable memory architectures in order to share data efficiently.

3.1.1 Maximum Clique and Upper Bounds

In maximal clique enumeration, it is often useful to first identify the size of a graph's maximum clique. Computing the maximum clique is important in a variety of biological applications, such as when establishing the edge-weight threshold in microarray analysis [20], when searching for common *cis* regulatory elements [11], and when solving the compatibility problem in phylogeny [30].

The maximum clique problem is a classic $\mathcal{NP}$-complete problem. From an asymptotic standpoint, the fastest known general-purpose maximum clique algorithms are based on independent sets and run in $O(2^{n/4})$ time, where n is the order of the input graph [64]. A better method is to reduce the clique problem to the vertex cover problem and to employ the notion of *fixed parameter tractability* ($\mathcal{FPT}$) on the vertex cover problem, where a *vertex cover* is a set of vertices that contains at least one endpoint of every edge. A problem is $\mathcal{FPT}$ if it has an algorithm that runs in $O(f(k)n^c)$ time, where n is the problem size, k is the input parameter, $f()$ is an arbitrary function, and c is a constant independent of both n and k [22]. When the input parameter k is fixed, $f(k)n^c$ is, simply, a polynomial. Thus an $\mathcal{FPT}$ problem is tractable even if the growth of $f(k)$ is very fast (e.g. factorial). $\mathcal{FPT}$ can be traced back to work done on applications of well-quasi order theory [27, 28, 29], where the main objective was to demonstrate, via the Graph Minor Theorem, that a variety of $\mathcal{NP}$-complete problems are actually tractable when the relevant input parameter is fixed.

Although it is known that the maximum clique problem is not in $\mathcal{FPT}$, the problem can be tackled by solving its complementary dual problem, the minimum vertex cover problem, which has been extensively studied as a $\mathcal{FPT}$ problem. The fastest known fixed parameter algorithm for vertex cover runs in $O(1.2759^k k^{1.5} + kn)$ time [16]. *Clique Enumerator* employs recent $\mathcal{FPT}$

implementations of vertex cover to solve the maximum clique problem [3, 2]. In these implementations, advanced kernelization techniques, such as linear programming and crown reduction, are first utilized to reduce the search space to a kernel; branching techniques are then used to traverse the search tree.

3.1.2 A k-Clique Enumeration Algorithm

Although neither *BK* algorithms [14] satisfy the requirement of finding cliques in order of size, they do provide a sound foundation for an algorithm to enumerate all cliques, maximal and non-maximal, of size k in canonical order. Both *Base BK* and *Improved BK* algorithms can be viewed as a depth-first traversal of a search tree, but *Improved BK* reduces the search space through a better candidate selection method. When choosing a vertex v to extend the clique, *Base BK* always chooses in the order as they were presented in the candidate set. In contrast, *Improved BK* initially chooses a vertex with the highest number of connections to the remaining members of the candidate set. After finding a maximal clique or returning from a child node, *Improved BK* only considers vertices that are not connected to the current v. This enables *Improved BK* to operate more efficiently on graphs with a high number of overlapping cliques [1].

However, the improvement in *Improved BK* was predicated on the need to find only maximal cliques. When finding both maximal and non-maximal cliques of a specified size, this constraint is no longer beneficial. Not selecting vertices connected to the current v upon finding a maximal clique or returning from a child node prevents the discovery of overlapping cliques of which at least one is non-maximal. In addition, given k, it is more efficient to eliminate all vertices of less than k-1 degree during preprocessing-such vertices cannot be members of any k-clique by definition-than to select a vertex of highest connectivity. For these reasons, *Base BK* was chosen as the basis for the algorithm presented here.

Algorithm 1 is a variant of the *Base BK* algorithm. Its core function `kclique_find()` finds not only maximal but also non-maximal cliques of a given size. Three vertex sets C, D, and N are used in the recursive function, where C contains the vertices that form a clique, D is the candidate set, and N is a set of vertices that have been previously added to C. This algorithm differs from the *Base BK* algorithm in two ways:

- *BK Base* examines the sets D, N, no matter what the size of C, and returns maximal k-cliques only when both D and N are empty. `kclique_find()` examines the sets D, N only when the size of C is equal to k and returns separately non-maximal k-cliques and maximal ones.
- A boundary condition is introduced in this algorithm. If at any point there are less than k vertices in the union of sets C and D, it is not possible to form a k-clique with the current C and D, thus `kclique_find()` returns.

It is worth noting that the preprocessing procedure that iteratively removes all vertices with degree less than k can greatly reduce the search space in practice because real graphs, including those from biological applications, are usually scale-free networks whose degree distributions follow a power law.

3.2 The MCLEARS Algorithm

A breadth-first search based algorithm, Maximal CLique Enumeration Algorithm for Ranged Sizes (MCLEARS), was developed to enumerate maximal cliques of size bounded by a given range in

Algorithm 1: The k-Clique Enumeration Algorithm

Input : a graph $G = (V, E)$ and an integer k
Output: a set L_k of non-maximal cliques of size k, with maximal k-cliques printed out

```
Preprocessing (G,k); // iteratively remove vertices of degree < k
procedure kclique_find (C, D, N);
C: a set of vertices that are the current clique;
D: a set of vertices that can be added to the current C;
N: a set of vertices that have been previously added to C;
begin
    if |C ∪ D| < k then return; // no enough vertices to form a k-clique
    if |C| = k // found a clique of size k
    then
        if D = ∅ and N = ∅ then PRINT(C); // report maximal clique
        else L_k ← L_k ∪ C; // store non-maximal clique
    else
        foreach u ∈ D do
            D ← D\{u};
            N[u] ← {v ∈ V|(u, v) ∈ E}; // the set of neighbors of vertex u in G
             kclique_find (C ∪ {u}, D ∩ N[u], N ∩ N[u]);
            N ← N ∪ {u};
end
```

non-decreasing order. The large memory requirement introduced by the breadth-first search is reduced by utilizing a method for checking maximality to avoid keeping all previously generated cliques in memory, a data structure for maintaining the list of cliques to avoid duplication of clique members, and a bitmap data representation for holding neighborhoods of cliques.

3.2.1 Algorithmic Basics

The MCLEARS algorithm was inspired by a characteristic property of cliques described in [46]: Any k-clique ($k \geq 3$) is comprised of two (k-1)-cliques that share (k-2) vertices. Figure 3.1 shows such an example. The property also implies the definition of maximal clique, any (k-1)-clique is maximal if it is not a subgraph of any k-cliques; otherwise, it is non-maximal.

The algorithm presented in [46] takes as input a list of all edges (2-cliques) in non-repeating canonical order, generates all possible (k+1)-cliques from all k-cliques, checks for all k-cliques to determine whether they are subgraphs of a (k+1)-clique after it is generated, declares a k-clique maximal if it is not a component of any (k+1)-cliques, outputs all the maximal k-cliques, and repeats this procedure until there is no (k+1)-clique generated. This algorithm provides two features that are not available in other methods, including the BK algorithms. The algorithm enumerates maximal cliques in non-decreasing order, and it prevents repetitive generation of non-maximal clique components in the search for maximal cliques [1]. However, the algorithm has the following disadvantages:

- It requires storage of all k-cliques and (k+1)-cliques, demanding an enormous amount of memory. Secondary storage such as disk scan be used, but the overhead of disk I/O operation is overwhelming.
- In order to decide if a k-clique is maximal, the algorithm must check for every (k+1)-clique to see if it contains any k-clique as its subgraph. This requires a search of all k-cliques, which makes the algorithm difficult to parallelize.

The MCLEARS algorithm proposed here enumerates maximal cliques in non-decreasing order by a breadth-first tree search. In addition, MCLEARS addresses the two issues noted above. MCLEARS decides the maximality of cliques without maintaining previously generated cliques. This feature not only reduces the amount of cliques that need to be stored, but also allows the generation of cliques independently, which enables efficient parallel implementations. Its application must observe the following:

Observation 1 A k-clique is a *candidate* to generate (k+1)-cliques if and only if it is a subgraph of a (k+1)-clique.

A k-clique is not a subgraph of any (k+1)-clique either if it is maximal, or it does not have (k-1) vertices shared with any other k-clique. Thus, it is not necessary to keep all k-cliques to generate (k+1)-cliques; only the candidate k-cliques are needed.

Observation 2 The k-1 vertices shared by two k-cliques is also a clique.

The k-cliques generated from the same (k-1)-clique naturally form a sub-list consisting of the (k-1)-clique with a list of neighbors common to this (k-1)-clique. The common neighbors of a clique are the vertices that are immediate neighbors to all vertices in this clique. Thus, when deciding whether a k-clique shares (k-1) vertices with any other k-clique it is simply necessary to decide

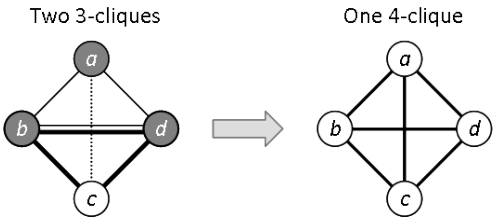


Figure 3.1: A k-clique ($\{a, b, c, d\}$) consists of two (k-1)-cliques ($\{a, b, d\}, \{c, b, d\}$) that share (k-2) vertices (b, d).

whether or not a k-clique is the only clique in a sub-list. If it is, it does not share (k-1) vertices with any other k-cliques. Moreover, to avoid the duplication of cliques, only the common neighbors whose indices are higher than the index of the (k-1)-th vertex need to be kept in a k-clique sub-list.

Observation 3 A clique is maximal if the clique has no common neighbors.

If a k-clique has any common neighbor, the k-clique together with any one of the common neighbors forms a (k+1)-clique. Thus, in order to check the maximality of a k-clique, it is not necessary to check all (k+1)-cliques to test whether the k-clique is contained in any of them, but only to confirm whether the clique has any common neighbors.

3.2.2 Bitmap Representation of Common Neighbors

The MCLEARS algorithm must compute the common neighbors of a k-clique to determine whether it is maximal. The procedure of finding the common neighbors of a clique can be solved by representing the common neighbors as a binary (bit) string and then applying bit operations on the string. The bit string representing the common neighbors of a clique is $\lceil n/8 \rceil$ bytes in length, where n is the number of vertices in the graph. Each bit represents the adjacency between the clique and a vertex that is not in the clique, i.e. the i-th bit is set to '1' if all vertices in the clique are adjacent to the i-th vertex and set to '0' otherwise. With such a representation, the common neighbors of a k-clique can be computed by bitwise AND operations (`BitAND`) of neighbors of the k vertices in the clique. The maximality of a clique can be determined by verifying whether a bit '1' exists (`BitOneExists`) in common neighbors of the clique. Table 3.1 shows an example of the bit strings of common neighbors of cliques in the graph shown in Figure 3.2 (b). The bit string of common neighbors of the 3-clique $\{a, b, c\}$ is 0001100, in which two '1' bits exist, thus clique $\{a, b, c\}$ is non-maximal. The bit string of the 5-clique $\{a, b, c, d, e\}$ is 0000000, which is all zeros, thus clique $\{a, b, c, d, e\}$ is maximal.

The common neighbors of a k-clique can be computed by either (k-1) `BitAND`s on neighbors of the k vertices, or one `BitAND` on common neighbors of (k-1)-clique and neighbors of a single vertex. The first method requires no additional memory, but it performs `BitAND` on the same bit strings repeatedly. The second method is faster, but it requires keeping the common neighbors of a (k-1)-clique in memory. The MCLEARS algorithm makes a tradeoff between these two methods. It maintains the common neighbors of the shared (k-1)-clique for each k-clique sub-list rather than for each k-clique, which avoids large memory requirement and repetitive bit operations. For each k-clique sub-list, the common neighbors of the (k-1) vertices shared by all k-cliques are kept. At that point only two `BitAND`s are necessary to compute the common neighbors of a (k+1)-clique. Taking the same example shown in Table 3.1, 3-cliques $\{a, b, c\}$ and $\{a, b, d\}$ sharing clique $\{a, b\}$ form a sub-list. The common neighbors of clique $\{a, b, c, d\}$ can be computed from a `BitAND` on

Table 3.1: An example of bit strings representing common neighbors of cliques in the graph G shown in Figure 3.2.

Vertex	Neighbors	Clique	Common Neighbors	Maximal?
a	0111100	$\{a, b\}$	0011100	No
b	1011110	$\{a, b, c\}$	0001100	No
c	1101111	$\{a, b, d\}$	0010100	No
d	1110101	$\{b, c, f\}$	0000000	Yes
e	1111001	$\{a, b, c, d\}$	0000100	No
f	0110001	$\{c, d, e, g\}$	0000000	Yes
g	0011110	$\{a, b, c, d, e\}$	0000000	Yes

neighbors of vertex d and common neighbors of $\{a, b, c\}$, which is computed from a `BitAND` on neighbors of vertex c and common neighbors of $\{a, b\}$.

3.2.3 Algorithmic Details

The MCLEARS is a breadth-first search based algorithm, thus it shares the same procedure as the algorithm in [46] but differs in two significant ways:

- When a k-clique is generated, MCLEARS directly checks its maximality by examining its common neighbors, instead of being stored and compared to all (k+1)-cliques subsequently generated;
- When generating (k+1)-cliques, only those candidate cliques are kept, instead of keeping all.

In MCLEARS, k-cliques that share (k-1) vertices form a k-clique sub-list. To generate (k+1)-cliques, the common neighbors of a k-clique sub-list (i.e. k-th vertex of each k-clique) are pairwise compared to check the adjacency. If two common neighbors are adjacent to one another, a new (k+1)-clique is generated. Those (k+1)-cliques generated from a same k-clique form a new (k+1)-clique sub-list. Each k-clique sub-list is deleted after its (k+1)-cliques are generated. When a (k+1)-clique is generated, whether or not it is maximal is determined by checking the clique for any common neighbors. The non-maximal (k+1)-cliques in those sub-lists that contain more than one non-maximal cliques are regarded as candidates. Algorithm 2 lists the pseudocode for the main function `GenerateKCliques()` that generates the list of candidate (k+1)-cliques and enumerates maximal (k+1)-cliques from a list of candidate k-cliques and the input graph.

Theorem 1 Given a graph G, a list of non-maximal lb-cliques and a size upper bound ub, the Maximal CLique Enumeration Algorithm for Ranged Sizes finds all maximal cliques of sizes within the range $[lb, ub]$.

Proof. The accuracy of MCLEARS is guaranteed by Observations 1, 2, and 3. MCLEARS performs a breadth-first traversal on the search tree. At each level of the tree, MCLEARS considers all the candidate k-cliques to expand them into (k+1)-cliques by Observation 2, checks their maximality by Observation 3, and removes only those (k+1)-cliques that do not lead to any larger cliques by Observation 1. In this way MCLEARS finds all maximal k-cliques and candidate k-cliques at each level of the search tree. Starting with the list of non-maximal lb-cliques, MCLEARS stops until all maximal ub-cliques are found. Upon termination, MCLEARS has found all maximal cliques of sizes within $[lb, ub]$.■

Algorithm 2: The Maximal Clique Enumeration Algorithm for Ranged Sizes (MCLEARS)

Input : A set L_k of candidate k-cliques, and a graph G
Output: A set L_{k+1} of candidate (k+1)-cliques, with the maximal cliques of size $k+1$ printed out

procedure `GenerateKCliques`(L_k, G);
S_k: set of k-cliques that share the same $k-1$ vertices;
C_k: a (k-1)-clique, is the set of $k-1$ vertices shared by S_k;
$N[u]$: the bit string of neighbors of vertex u in G;
$N[C_k]$: the bit string of the common neighbors of a clique C_k in G;

begin
  $L_{k+1} \leftarrow \emptyset$;
  for *each* $S_k \in L_k$ **do**
    $C_k \leftarrow$ the $k-1$ vertices shared in S_k; `// Observation 2:expand clique`
    for $i \leftarrow 1$ **to** $|S_k| - 1$ **do**
      $v \leftarrow k$-th vertex of i-th clique $\in S_k$;
      $S_{k+1} \leftarrow \emptyset$;
      $C_{k+1} \leftarrow C_k \cup \{v\}$;
      $N[C_{k+1}] \leftarrow$ `BitAND(`$N[C_k]$, $N[v]$`)`;
      for $j \leftarrow i+1$ **to** $|S_k|$ **do**
        $u \leftarrow k$-th vertex of j-th clique $\in S_k$;
        if $(v,u) \in E(G)$ **then**
          if `BitOneExists(BitAND(`$N[C_{k+1}]$, $N[u]$`))` $= TRUE$ **then**
            $S_{k+1} \leftarrow S_{k+1} \cup \{C_{k+1} \cup \{u\}\}$; `// non-maximal clique`
          else
            `PRINT(`$C_{k+1} \cup \{u\}$`)`; `// Observation 3:maximal clique`
      if $S_{k+1} > 1$ **then**
        $L_{k+1} \leftarrow L_{k+1} \cup \{S_{k+1}\}$; `// Observation 1:candidate cliques`
  return L_{k+1};
end

An Example

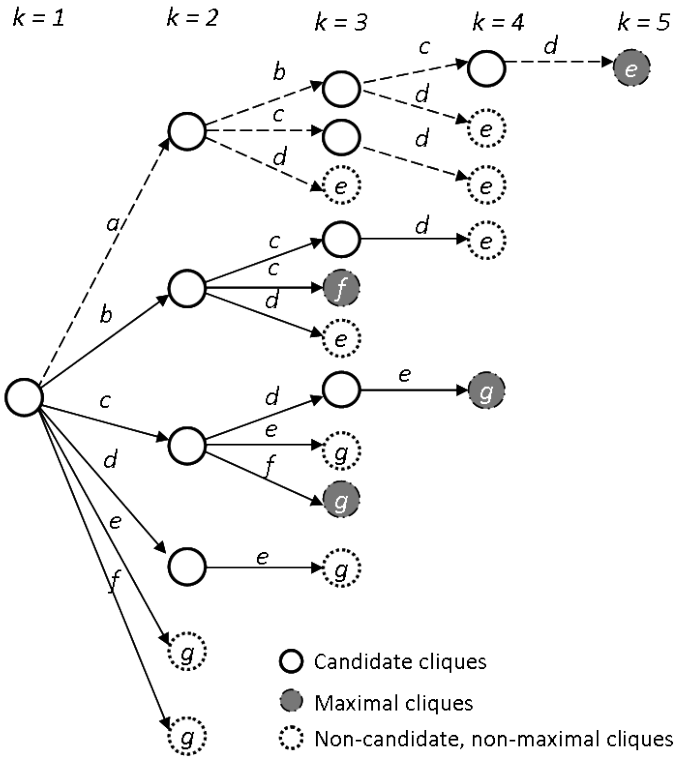
Figure 3.2 illustrates the procedure of the MCLEARS algorithm on graph G with two maximal 3-cliques, one maximal 4-clique, and one maximal 5-clique. In the search tree (Figure 3.2 (a)), the vertices on the path starting at the root form a clique. The leaf nodes in gray are maximal cliques formed by the vertices on its path with the vertex labeled in the leaf node. The intermediate nodes are candidate cliques. The leave nodes in dashed line are non-maximal, non-candidate cliques. Figure 3.2 (d) illustrates the procedure of the MCLEARS algorithm on one branch of the search tree (those in dashed arrows). Common neighbors of a k-clique (i.e. nodes in light gray) sub-list are compared to other common neighbors with higher indices to generate (k+1)-cliques. Cliques are generated in non-repetitive canonical order. For example, from the 2-clique sub-list that shares vertex a, the first common neighbor vertex b is compared to c, d, and e respectively to get three 3-cliques $\{a, b, c\}$, $\{a, b, d\}$ and $\{a, b, e\}$. Vertex c, the second common neighbor in the list, is compared only to d and e, whose indices are higher to avoid duplication. Similarly, common neighbor d will be compared to e, and the last common neighbor e will be skipped since it has been compared to all other common neighbors. The generation of (k+1)-cliques from one k-clique sub-list is independent of any other k-clique sub-lists. For example, the 3-cliques generation from the 2-clique sub-list sharing vertex a and that from the 2-clique sub-list sharing vertex b are independent of one another.

The MCLEARS algorithm has the following advantages. First, it enumerates maximal cliques in non-decreasing order. Second, it keeps only the candidate k-cliques to generate (k+1)-cliques and deletes a k-clique sub-list once (k+1)-cliques are generated from it. Third, the bitmap representation of common neighbors allows fast bit operations to decide whether or not a clique is maximal. The procedure to decide whether a clique is maximal is to check for the existence of bit '1' in a bit string of length n. Finally, the algorithm is parallel because the generation of (k+1)-cliques from one k-clique sub-list is independent of any other k-clique sub-lists, and the procedure to check whether a clique is maximal is independent of any other cliques.

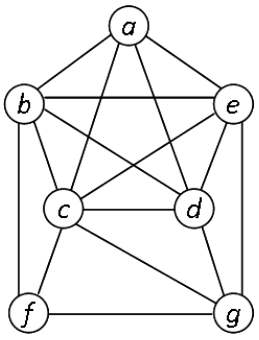
3.2.4 Algorithmic Complexity

Given a graph $G = (V, E)$, let $n = |V|$, and $m = |E|$. Let N_k be the number of candidate k-clique sub-lists, and M_k be the total number of candidate k-cliques. Initially, $N_2 \leq n - 2$ and $M_2 = m$, because only the first $n-2$ vertices are possible to generate 2-clique sub-lists containing more than one clique, and there are only m edges. At each step k, assuming N_k and M_k are given, then $N_{k+1} \leq M_k - 2N_k$, because all candidate k-cliques, except those which are the last two in a k-clique sub-list, are able to generate (k+1)-clique sub-lists that contain more than one clique. Each (k+1)-clique sub-list contains at most $n - k$ cliques. We then have $M_{k+1} \leq (M_k - 2N_k)(n - k)/2$ because only the common neighbors with higher indices are compared to generate (k+1)-cliques, which results in a total number that is half of the product.

For each k-clique sub-list, the run time for checking adjacency of pairs of common neighbors is $O((n - k)^2)$, and checking whether a k-clique is maximal takes $O(n)$ (in the worst instance). Therefore, the run time of step k is $O((n-k)^2 N_k + nM_k)$. As for the space requirement, assuming each vertex index takes c bytes, then at each step k the algorithm would need $cM_k + (c(k-1) + n/8)N_k$ bytes to hold all the candidate k-cliques, and N_k*`sizeof(pointers)` more bytes to keep the pointers to the sub-lists.



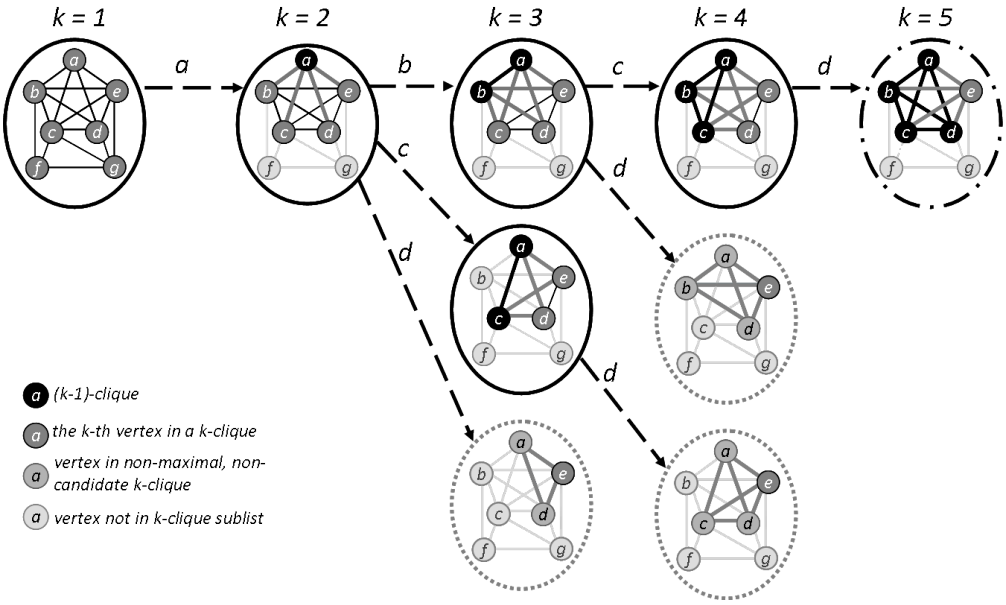
(a) The breadth-first search tree T_1.



(b) A graph G.

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	1	1	1	1	0
c	1	1	0	1	1	1	1
d	1	1	1	0	1	0	1
e	1	1	1	1	0	0	1
f	0	1	1	0	0	0	1
g	0	0	1	1	1	1	0

(c) Its adjacency matrix.



(d) A branch (in dashed arrows) of the search tree T_1.

Figure 3.2: An example of the MCLEARS algorithm applied to a graph: The search tree T_1 of the algorithm (a) on graph G (b), and a branch of the search tree T_1 (d). In each node of the branch, the vertices in gray are candidate k-cliques with its (k-1)-clique in black. The vertex identifier labeled in each leaf node in T_1 is the last vertex in a clique formed by the vertices on the path from the root.

3.3 A Parallel Enumeration Algorithm on Shared-Memory Machines

The MCLEARS algorithm reduces the storage requirement by maintaining only the cliques that can produce larger maximal cliques and by using the bitmap representation. However, the number of candidate cliques can still be too large to be stored in the memory of a single computer. To solve larger problems in a reasonable period of time, the MCLEARS algorithm is parallelized on NUMA scalable shared memory multi-processor (S^2MP) machines.

3.3.1 The NUMA S^2MP Architecture

The Non-Uniform Memory Architecture (NUMA) is a memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. In NUMA, a processor accesses its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors.

The memory layout of the NUMA S^2MP architecture is shown in Figure 3.3. In this layout, the smallest system is a single node. Each node consists of two processors, memory, and a device called "the hub" that manages each processor's access to memory and I/O. Larger systems are built by connecting multiple nodes. A two-node system is formed by either simply wiring the two hubs together or by employing a router to connect the two hubs. Figure 3.3 shows an eight-node, or an equivalently sixteen-processor, system that is constructed from four two-node with router building blocks. The key feature of this architecture is that the hardware allows the physically distributed memory of the system to be shared, just as in a bus-based system; however, because each hub is connected to its own local memory, the access time to memory is no longer uniform. Accessing remote memory through an additional hub adds an extra increment of time, as does each router that the data must travel through.

3.3.2 Requirements for Parallelism

The multi-threaded implementation of the parallel algorithm is based on the fact that the generation of $(k+1)$-cliques from a k-clique sub-list is independent of any other k-clique sub-lists. However, the approach is not as easy as it sounds. There are three requirements to be considered during the design:

- The desired feature of the MCLEARS algorithm that enumerates cliques in non-decreasing order must be maintained. As a result, multiple threads must be synchronized to generate cliques of the same size;
- The computation workloads must be balanced among threads because the number of cliques generated could vary greatly from one sub-list to another;
- Each thread is expected to work on its local instance as much as possible to avoid remote memory access.

3.3.3 The Task Scheduler

To meet the requirements, a master/worker programming model is chosen for synchronization and load balancing, as illustrated in Figure 3.4. The master thread (task scheduler) is responsible for

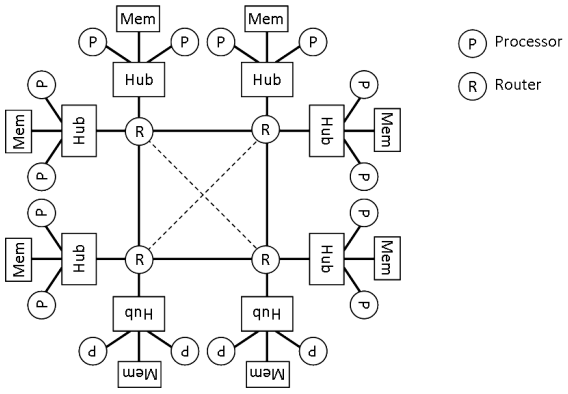


Figure 3.3: The memory layout of NUMA scalable shared memory multi-processor architecture

synchronizing multiple worker threads, initially distributing tasks, making load-balancing decisions, and collecting partial results.

- Worker threads are synchronized by the task scheduler to work simultaneously on the same level of the search tree. This is multi-stage synchronization, where each level is called a "stage";
- The task scheduler makes load transfer decisions at the beginning of each stage to ensure that the workers have roughly the same degree of workload. We choose not to initiate load balancing during the stages because a thread working on loads transferred from other threads must access the remote memory on that processor, thus introducing extra overhead;
- To avoid too much remote memory access, the task scheduler prioritizes the dual threads to be the source and destination of workload transfer decisions, where dual threads are defined as those on the dual processors sharing the same physical memory slot.

In parallel MCLEARS, a work unit is a k-clique sub-list, and the workload of a work unit is the number of k-cliques it contains. A worker thread may have more than one work unit at each level. The workload of a worker is the total number of k-cliques it possesses.

Initially, when multiple worker threads fork off, the task scheduler explores the first level of the search tree, evaluates the workload, and evenly distributes the work units to the worker threads. Each worker thread should have the same workload to start with. Workers then start to generate cliques independently since the traversal of branches on the search tree are independent of one another. When a worker finishes its work in the current stage, it sends back the statistical results of this stage and the workloads status for the next stage to the task scheduler, and waits for the synchronization signal to begin the next stage. The task scheduler must determine whether it is necessary to initiate load balance and, if so, to identify the source and destination of the work transfer and to calculate how much work should be transferred. Once a load balance decision is made and the work transfer is finished, the task scheduler signals all workers to start the next stage. The two core functions `Task_Scheduler`() and `Worker_Thread`() are given in Algorithm 3.

There is no communication among threads when the load balancing is performed because the large data structure is shared in the memory. This is why shared-memory machines are preferable to distributed-memory machines in our implementation. The task scheduler must be adaptable when it makes a decision to transfer tasks among workers. A dynamic load balancing technique is thus used in the task scheduler to transfer work from heavily loaded workers to lightly loaded (or

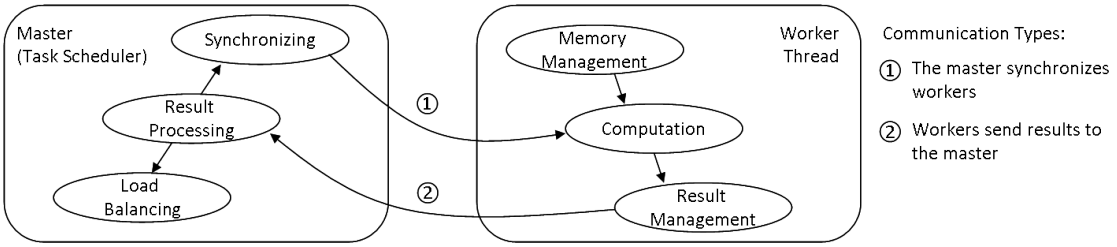


Figure 3.4: The master/workers model for synchronization and load balancing in the parallel algorithm for maximal clique enumeration

Algorithm 3: The Parallel MCLEARS Algorithm

Input : A set L of candidate cliques of size k, and a graph G
Output: Maximal cliques of size $x > k$

procedure `Task_Scheduler`(L, G, k);
begin
  divide L evenly to worker threads $\{S_1, S_2, ..., S_{np}\}$;
  while *there are candidate k-cliques* **do**
    signal worker threads to start;
    while *there are workers still working* **do**
      collect the statistical results from worker threads;
    make the load balancing decision and redistribute the work;
    k++;
  signal all worker threads to quit;
end

procedure `Worker_Thread`(S, G, k);
S: a subset of L;
begin
  while *True* **do**
    wait for signal;
    if the signal is to quit **then break**;
    $S' \leftarrow$ `GenerateKCliques(`S, G`); // maximal cliques are printed out`
    send the statistical results to the task scheduler;
    k++;
end

idle) ones. The scheduler computes the average workload and the differences between the workloads of each worker and the average workload. If the workload of a worker is lower than average and the difference is greater than the threshold, it is designated as a "light-loaded worker." The heavy-loaded workers are similarly defined. The threshold is determined based on the graph size, the total number of workloads, and the average and standard deviation of workloads. The scheduler makes a work transfer decision when there are workers with light and heavy workloads. When the scheduler chooses the source and destination of the work transfer, the dual threads are prioritized to balance one another's workload.

An Example

Figure 3.5 shows how the load balancing strategy works. In this example, there are two worker threads w_1, w_2. The inputs are six (k-1)-clique sub-lists that contain (4, 4, 4, 2, 1, 1) (k-1)-cliques. At k level, the scheduler initially distributes the six work units evenly to the workers; worker w_1 is assigned two work units, and w_2 is assigned four work units, each having a total workload of 8. At k+1 level, w_1 has five work units of a total load of 9, while w_2 has four work units of a total load of 5. The task scheduler then makes a load balancing decision to transfer one work unit of load 2 from w_1 to w_2. As a result, both workers have workload of 7. Similarly, at k+2 level, both workers have a workload of 3. Thus the scheduler decides that no load balancing is needed at this level.

3.3.4 Memory Management

Memory management is an important issue when implementing the parallel algorithm on the shared-memory architecture since the memory under NUMA is physically distributed. To efficiently utilize the local memory for each thread, a memory management module is implemented to allocate and free dynamic memory. Functions (e.g. `fmalloc()`, "faked malloc") are implemented to replace the `libc` memory management routines (e.g. `malloc()`). Initially, each thread allocates as much local memory as possible from the system. After that, the program manages the memory using its own functions. By implementing a memory management module, the worker thread secures the local memory for its use. Furthermore, it avoids fragmented memory when requesting dynamic memory allocation each time a new node is created on the search tree.

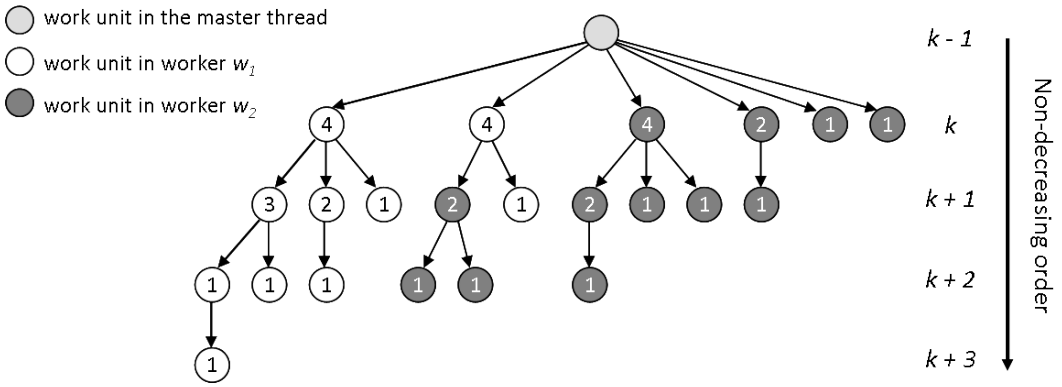


Figure 3.5: The load balancing strategy in parallel MCLEARS for maximal clique enumeration. Each node on the tree represents a work unit. The number in each node represents the workload. The white nodes are work units assigned to worker w_1, and the dark gray nodes are those assigned to worker w_2.

3.4 Performance Evaluation

In this section, the performance of the MCLEARS algorithms, both the sequential and parallel versions, are evaluated. The amount of memory required by the parallel algorithm is also measured. The implementation of the MCLEARS algorithm ran on a 1 GHz Mac PowerPC G4 with 1GB memory to compare its relative efficiency with other algorithms. The tests of the parallel algorithm are performed on an SGI Altix with 256 Intel Itanium 2 processors running at 1.5 GHz and 8 GB of memory per processor for a total of 2 Terabytes shared system memory.

The graphs used in our tests were generated from raw microarray data after normalization, pairwise rank coefficient calculation, and filtering using a threshold [10]. Two of the graphs were generated from neurobiological data sets, each containing 12,422 vertices, one with 6,151 edges (0.008% edge density), the other with 229,297 edges (0.3% edge density) [20]. The third graph was generated from myogenic differentiation data containing 2,895 vertices with 10,914 edges (0.2% edge density) [69]. Applying *Clique Enumerator* to these graphs, the maximum clique sizes are found to be 17, 110, and 28 for each graph, respectively. Each experiment was repeated 10 times and the average run time was reported.

3.4.1 Results on Sequential Algorithms

The performance of the MCLEARS algorithm was compared against an implementation of the algorithm in [46], which provides the same feature as MCLEARS that finds cliques in nondecreasing order of size. The implementation stores cliques in memory and thus is called *Kose RAM* [1]. Table 3.2 presents run times for *Kose RAM* and the sequential MCLEARS algorithm that enumerate maximal cliques ranging in size from 3 to 17 on the 12,422 vertex graph. MCLEARS is more than three hundred times faster than *Kose RAM* due to the fast bit operations and reduced candidate cliques to be checked.

3.4.2 Results on the Parallel Algorithm

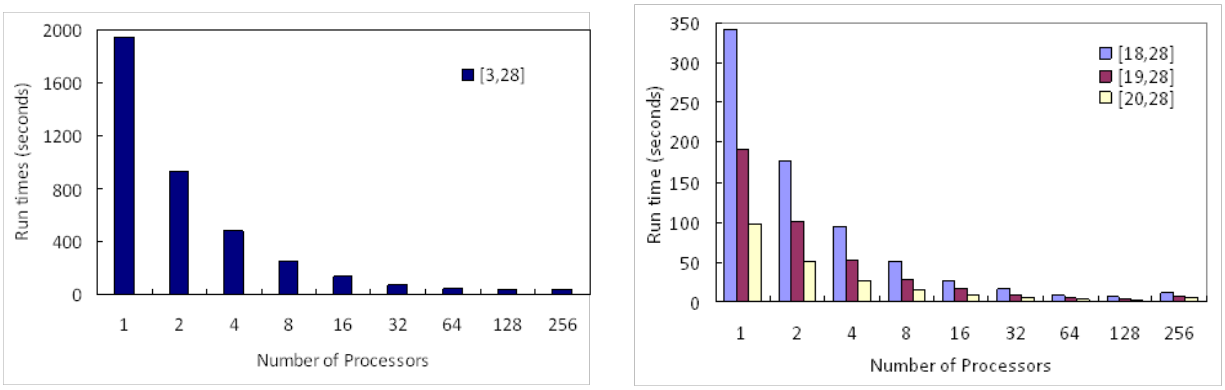
Figure 3.6 shows the run times of the multithreaded implementation with load balancing to enumerate maximal cliques within different size ranges on the 2,895 vertices graph using up to 256 processors on an SGI Altix 3700. The run times are the average of 10 different runs and the standard deviations are less than 5%. The variation of run times came from memory access times due to the different locations of memory allocated by system each time. The figure shows that the run times scale well for up to 64 processors and still scale when using 128 processors. Another observation is that the performance almost doubles when the lower bound of clique size is increased by one. This indicates that the run times of the algorithm largely depend on the clique size range, which decides the number of cliques in a graph.

Two speedup measures were also calculated to evaluate parallelization efficiency: the *absolute speedup*, defined as the ratio between p processors and one processor run times, and the *relative speedup*, defined as the ratio between $2p$ processors and p processors run times. The absolute and relative speedups for up to 64 processors are plotted in Figure 3.7. It shows that the relative speedups remain around 1.8 when the number of processors increases. This performance pattern is observed for all different clique size lower bounds from 3 to 20, though the absolute speedups for case range $[3, 28]$ are better than the absolute speedups for the other three cases.

The multithreaded program does not scale on 256 processors. The lack of scalability for 256 processors is largely due to the small execution time of the serial implementation, which becomes almost negligible when divided among 256 processors and is dominated by network and synchronization latency. As shown in Figure 3.8 (a), the absolute speedup for 256 processors increases when

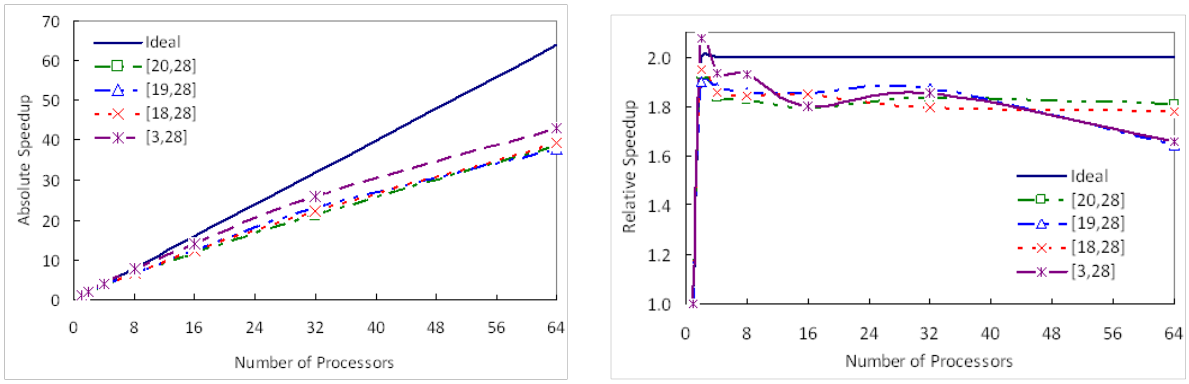
Table 3.2: Wallclock Run times of maximal clique enumeration by the sequential MCLEARS algorithm and *Kose RAM*

Graph Size	Edge Density	Maximal Clique Size	Total number of Maximal Cliques	*Kose RAM* (seconds)	*MCLEARS* (seconds)	Speedup
12,422	0.008%	[3, 17]	5,227	17,261	45	383



(a) (b)

Figure 3.6: Average run times to enumerate maximal cliques for all clique sizes (a) and different clique size ranges (b) for the 2,895 vertices and 0.2% edge density graph using up to 256 processors on an SGI Altix 3700.



(a) (b)

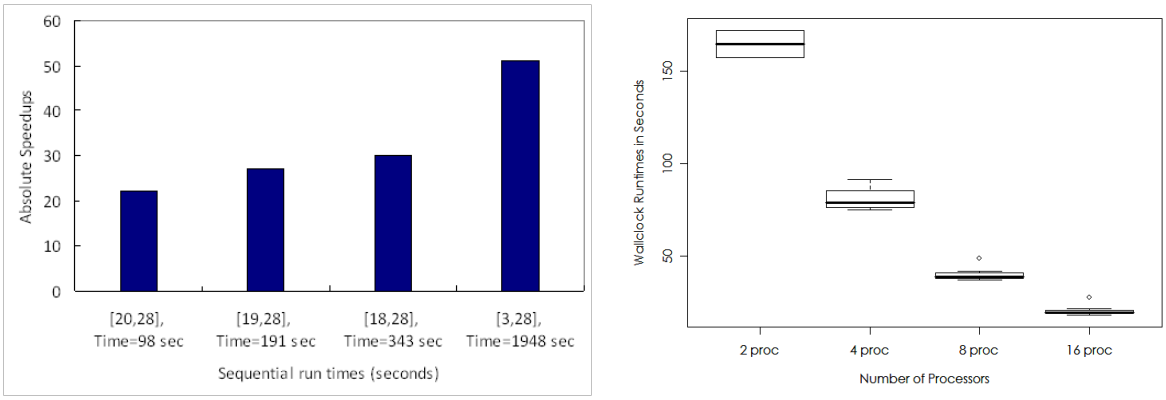
Figure 3.7: Absolute speedups (a) and relative speedups (b) for different clique size ranges using up to 64 processors.

the sequential run time increases. The speedup goes up from 22 to 51 when the sequential run time increases from 98 seconds for range $[20, 28]$ to 1,948 seconds for range $[3, 28]$. Similar behavior is observed for 128 processors. In other words, various problem sizes with different execution times have their own optimal number of processors to acquire the maximum benefit from the parallel algorithm.

The mean and standard deviation of the execution time across different processors are also plotted on the 2,895 vertices graph with range $[18, 28]$ in Figure 3.8 (b) to show the load balancing of the program. The standard deviations are within 10% of the average run times, indicating that the loads are well balanced across multiple processors during execution. Only 2 to 16 processors are plotted here because the mean and standard deviation become a single point starting from 32 processors.

3.4.3 Results on Memory Requirements

All cliques from size 3 to the maximum size are enumerated for denser graphs to understand the memory requirements. Figure 3.9 shows the memory used to keep all cliques of different sizes during the procedure of clique enumeration on the graph with 2,895 vertices. It roughly follows a normal distribution. Memory use first increases with clique size and goes up to almost 20 GB when clique size reaches 13, then it begins to drop quickly. When the algorithm was tested on the 12,422 vertices denser graph and terminated after 12 hours, the program had consumed 607 GB memory to hold newly generated $(k+1)$-cliques and 404 GB to hold k-cliques. Thus, the algorithm takes advantage of the shared memory architecture. Also, if the chosen lower bound of clique size is large enough to avoid the greatest memory requirement region and small enough still to be interesting for the application, MCLEARS would be able to enumerate maximal cliques within the range efficiently.



(a) (b)

Figure 3.8: (a) Absolute speedups for different initial clique sizes using 256 processors versus different sequential execution times. (b) Load balancing of run times across multiple processors on the 2,895 vertices graph with clique size range [18,28] using up to 16 processors.

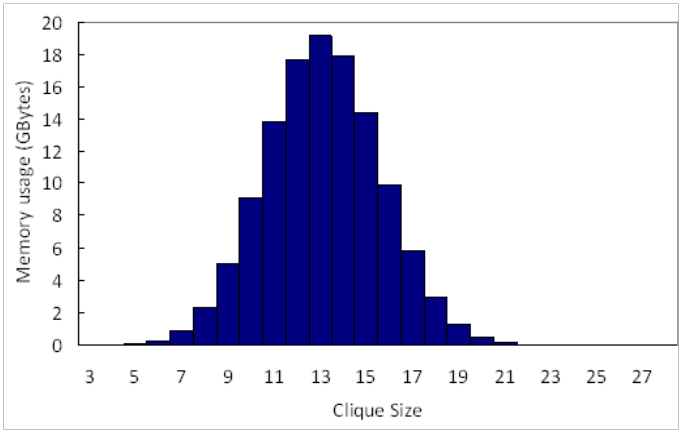


Figure 3.9: Memory usage in gigabytes on a SGI Altix for enumerating cliques of different sizes for the 2,895 vertices and 0.2% edge density graph.

Chapter 4

Algorithms for Bipartite Graphs

The integration of multiple genome-scale data sets is an algorithmic challenge for modern systems biology. In such settings, bipartite graphs are often useful in representing relationships across pairs of heterogeneous data types, with the interpretation of such relationships accomplished through an enumeration of maximal bicliques. Unfortunately, existing algorithms are highly inefficient and do not scale for this important task.

In this chapter, a fast and novel maximal biclique enumeration algorithm (MBEA) is described. It finds all maximal bicliques in a bipartite graph. Unlike other techniques that have been proposed for this problem, MBEA neither places undue restrictions on inputs and outputs nor inflates the problem size. Efficiency is achieved by exploiting structure inherent in bipartite graphs and by ensuring that biclique enumeration avoids duplication while pruning non-maximal candidates. Experiments on bipartite graphs generated from both biological data and synthetic data indicate that MBEA can be as much as two to three orders of magnitude faster than the best known alternatives.

4.1 The MBEA Algorithm

The MBEA algorithm is a backtracking method that uses a branch-and-bound technique to prune branches that cannot lead to a maximal biclique. It is inspired by the BK algorithm [14] for finding all maximal cliques of an undirected graph. The search space for MBEA algorithm is limited to disjoint vertex sets due to the fact that in a biclique, vertices in one set determine the vertices in the other. The algorithm operates on the set with the fewer number of vertices, which leads to a smaller search space.

4.1.1 Algorithmic Basics

Given a bipartite graph $G = (U \cup V, E)$ without loss of generality, we assume that $|U| \geq |V|$. MBEA utilizes four dynamically changing collections of vertices:

1. R, a subset of V,
2. L, a subset of U containing all the common neighbors of R,
3. P, a subset of V containing candidate vertices that may be added to R, and
4. Q, a subset of V containing former candidates, that is, vertices that were previously considered for R.

The sets R, L, P and Q are employed in a depth-first traversal of a recursion tree to form maximal bicliques. R and L are used to form such a biclique, where R determines L. P is used for biclique expansion. Q is used to determine maximality. P, Q, and R are required to satisfy the following:

- $(P \cap Q) \cup (P \cap R) \cup (Q \cap R) = \emptyset$. That is, P, Q, R are pairwise disjoint.
- $P \cup Q = \{v \mid v \in V \backslash R, \exists\ u \in L, (u, v) \in E\}$. That is, P and Q contain every vertex in V but not R that is adjacent to at least one vertex in L.

Observation 4 The subgraph induced by (L, R) is a biclique.

For simplicity, the reference to induced subgraph is dropped so that (L, R) is a biclique. Note that (L, R) is maximal iff there is no vertex in $U \backslash L$ that is adjacent to all vertices in R, and no vertex in $V \backslash R$ that is adjacent to all vertices in L. Because L is defined by R, only the maximality of R need be considered.

Observation 5 (L, R) is maximal iff no vertex in $V \backslash R$ is adjacent to every vertex in L.

If P contains a vertex that is adjacent to all vertices in L, then (L, R) is not maximal. That vertex may as well be moved from P to R. This process can be iterated until no more vertices can be so moved. On the other hand, if there is no vertex in $V \backslash R$ that is a common neighbor of all vertices in L, then (L, R) is maximal because L and R are the largest set of common neighbors of each other.

Observation 6 Let S denote $\{v \mid v \in P \text{ and } (u, v) \in E\ \forall u \in L\}$. Then $(L, R \cup S)$ is a maximal biclique.

If Q contains a vertex that is adjacent to all vertices in L, then not only (L, R) is not maximal, but there can be no S as defined above for which $(L, R \cup S)$ is maximal. This can be stated as follows:

Observation 7 Let T denote $\{v \mid v \in Q \text{ and } (u, v) \in E\ \forall u \in L\}$, let L' denote any subset of L, and let S' denote any subset of P. Unless T is empty, $(L', R \cup S')$ is not a maximal biclique.

Observation 7 is used to prune unproductive branches in a branch-and-bound style exploration of the maximal biclique search space. As Observation 5 shows, if Q contains a vertex v adjacent to all vertices in L, it means that biclique (L, R) is not maximal. It can also be observed that none of the bicliques extended from R contains v since R doesn't contain v. However, v is adjacent to all vertices in any subset of L. Thus, no bicliques extended from such a node is maximal and its branches can hence be pruned away.

4.1.2 Algorithmic Details

MBEA is a depth-first traversal of the recursion tree whose core is a recursive function `biclique_find()` as shown in Algorithm 4. R and Q are initially empty. All vertices in U and V are initial candidates. At each node of the search tree, taking as inputs a 4-tuple $\langle L, R, P, Q \rangle$, the function `biclique_find` first selects a candidate x from the set P. It then creates a new set R' from the old set R with x, which is the extension step. Subsequently, `biclique_find` creates a new set L' from the old set L by removing all vertices not connected to the selected candidate x, which makes L' a set of

common neighbors of vertices in R'. The next step is to create new sets P' and Q' from the old sets by removing all vertices that are not connected to any vertex in the new set L', which leaves only those vertices that are adjacent to at least one vertex in L' in the new P' and Q' sets. Moreover, `biclique_find` also removes from the new set P' vertices connected to all vertices in new set L and adds them to the set R, based on Observation 6. After that, if no vertex in set Q' is connected to all vertices in new set L', a maximal biclique with new sets (L', R) is found according to Observation 5. Finally, `biclique_find` is called recursively to operate on the new 4-tuple $\langle L', R', P', Q' \rangle$. Upon return, it removes the selected candidate x from P and adds it to the old set Q. The branch stops when the candidate set P is empty or a vertex in Q is found connected to all vertices in L because of Observation 7.

Initially, all vertices in V are candidates ($P = V$); the biclique is empty but all vertices in U are candidates ($R = \emptyset, L = U$); and there is no former candidate ($Q = \emptyset$). On the way from the top level to the bottom level of the recursion tree, the size of R keeps increasing and the sizes of L and P keep decreasing. And on the same level of the recursion tree, the size of Q on the right sibling is larger than that of its left sibling.

Theorem 2 Given a bipartite graph G, the Maximal Biclique Enumeration Algorithm finds all maximal bicliques of graph

Proof. The accuracy of MBEA is guaranteed by Observations 4, 5, 6 and 7. MBEA explores the entire search space of all the subsets of one vertex set, finds all the bicliques by Observation 4, checks their maximality by Observation 5, and skips or cuts off only those do not lead to any maximal bicliques based on Observations 6 and 7. Therefore, upon termination, MBEA has found all maximal bicliques. ■

An Example

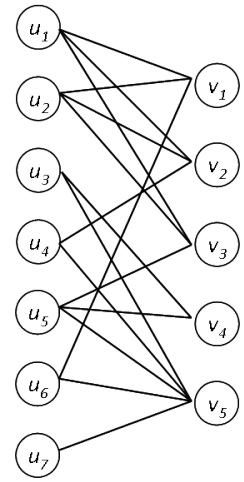
Figure 4.1 is an example of the recursion tree of MBEA for a bipartite graph G with $|U| = 7, |V| = 5, |E| = 16$ (Figure 4.1 (a)). The edges of the tree were labeled by the vertex selected from the candidate set. Each node on the tree is a biclique, formed by the vertices in V on its path from the root with their common neighbors in U. On the recursion tree, nodes in gray are maximal, nodes in dashed circles are not, and the root node is in white. For instance, the leftmost branch examined two candidate bicliques $\{v_1\}$ and $\{v_1, v_2, v_3\}$ on its path, and both are maximal bicliques. From the recursion tree T_2 shown in Figure 4.1 (b), it can be seen that a total of 12 nodes, excluding the root, are searched by MBEA for this bipartite graph. Among them, 9 are maximal bicliques and 3 are non-maximal bicliques. Note that exhaustive search will explore $2^5 = 32$ nodes. Furthermore, the detailed recursion tree shown in Figure 4.1 (c) illustrates the procedure of the algorithm. In each node, the sets L, R, P, and Q are colored using different gray scales in the graph G. The leaf node on path v_1, v_3 is not maximal according to Observation 7 because one vertex v_2 in its former candidates set Q is adjacent to the all vertices u_1, u_2 in its L.

4.1.3 Algorithmic Complexity

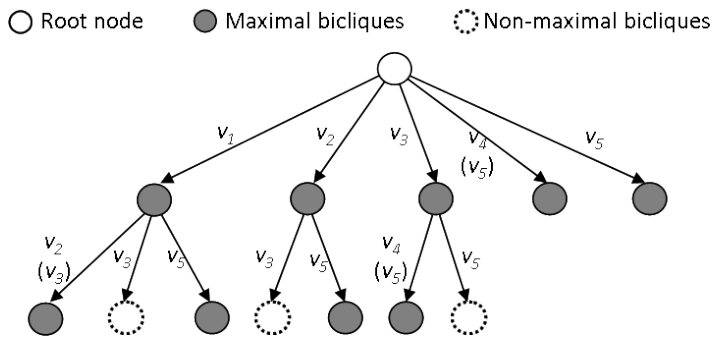
One case to consider is the time complexity of a simple brute-force algorithm that examines all subsets of the smaller vertex set. Given a bipartite graph $G = (U \cup V, E)$ where $|U| = m, |V| = n$, and $m \geq n$, the total number of subsets of V is 2^n. For each subset of V, the time to get the common neighbors in U is $O(nm)$, and the time to decide the maximality of the biclique formed by the subset with its common neighbors is also $O(nm)$. Thus, the worst-case time complexity of the brute-force algorithm is $O(nm \cdot 2^n)$.

Algorithm 4: The Maximal Biclique Enumeration Algorithm (MBEA)

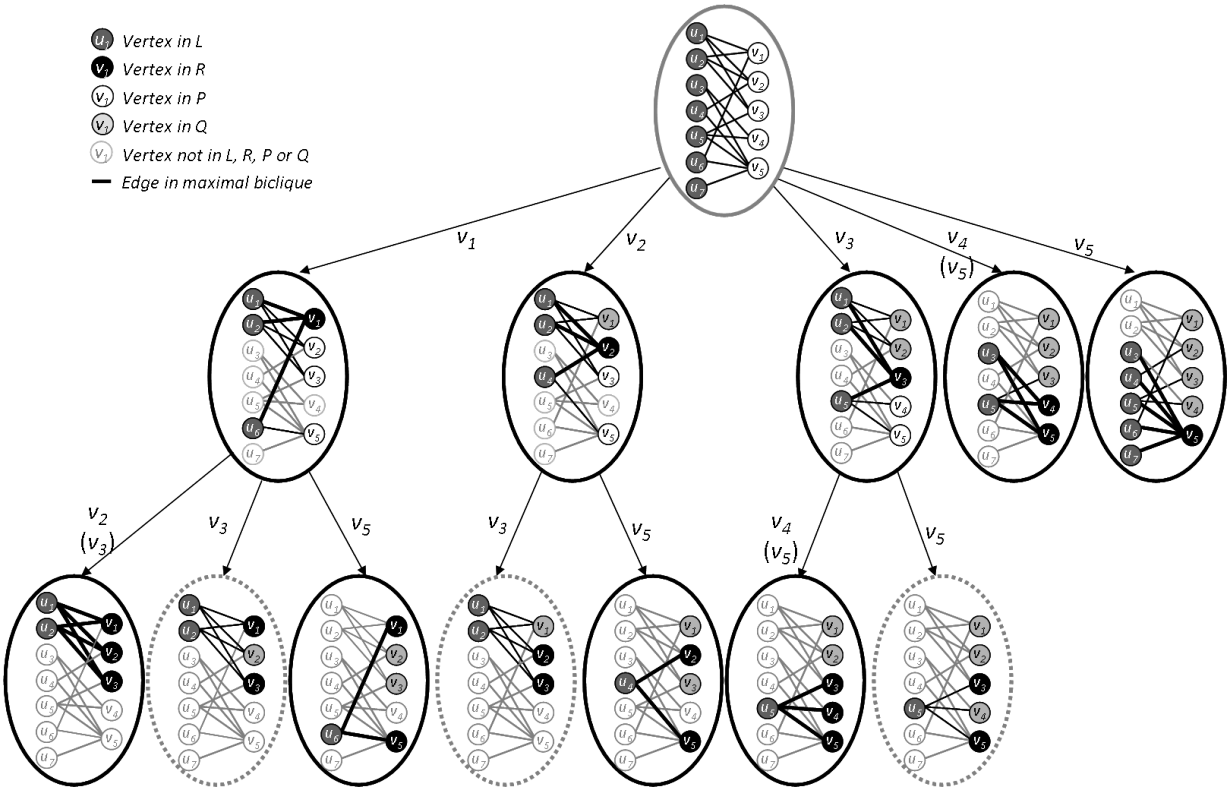
```
procedure biclique_find(G, L, R, P, Q);
G: a bipartite graph G = (U ∪ V, E);
L: set of vertices ∈ U that are common neighbors of vertices in R;
R: set of vertices ∈ V belonging to the current biclique;
P: set of vertices ∈ V that can be added to R;
Q: set of vertices ∈ V that have previously been added to R;
while P ≠ ∅ do
    Select x from P;
    P ← P\{x};
    // Observation 4:extend biclique
    R' ← R ∪ {x};
    L' ← {u ∈ L | (u, x) ∈ E(G)};
    // Create new sets
    P' ← ∅; Q' ← ∅;
    // Observation 5:check maximality
    is_maximal ← TRUE;
    forall v in Q do
        N[v] ← {u ∈ L' | (u, v) ∈ E(G)};
        // Observation 7:end of branch
        if |N[v]| = |L'| then
            is_maximal ← FALSE;
            break;
        else if |N[v]| > 0 then Q' ← Q' ∪ {v};
    if is_maximal = TRUE then
        forall v in P, v ≠ x do
            N[v] ← {u ∈ L' | (u, v) ∈ E(G)};
            // Observation 6:expand to maximal
            if |N[v]| = |L'| then R' ← R' ∪ {v};
            else if |N[v]| > 0 then P' ← P' ∪ {v};
        PRINT(L', R'); // Report maximal biclique
        if P' ≠ ∅ then biclique_find(G, L', R', P', Q');
    // Move x to former candidate set
    Q ← Q ∪ {x};
```



(a) A bipartite graph G;



(b) A recursion tree T_2 for G;



(c) The detailed recursion tree T_2.

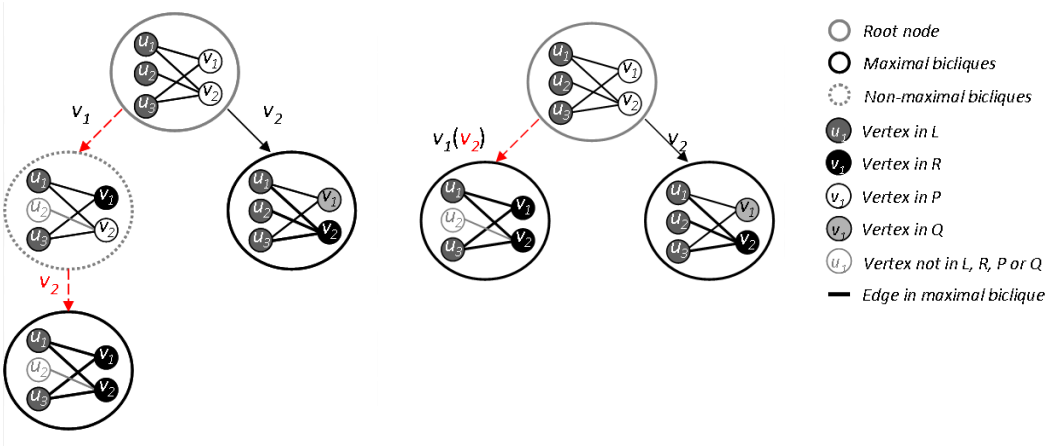
Figure 4.1: The recursion tree T_1 (b) of the algorithm MBEA for a bipartite graph G with $|U| = 7, |V| = 5$ (a). Nine maximal bicliques are found by searching on the recursion tree a total of 12 nodes excluding the root. The detailed recursion tree T_2 (c) illustrates the procedure of the MBEA algorithm on G by coloring the vertices in $\langle L, R, P, Q \rangle$ with different gray scales. The vertices in parentheses on the paths are those added by Observation 6.

In the worst case, the number of nodes on a recursion tree of the MBEA algorithm is the total number of subsets of the smaller vertex set. At each node, the time complexity of the recursive function `biclique_find` is $O(dn)$ where d is the maximum degree of vertices in V because the number of vertices in P and Q is at most n, and the number of vertices in L is at most d. Therefore, the worst case time complexity of MBEA with respect to the number of vertices is $O(dn \cdot 2^n)$. However, because of the branch-and-bound technique used in MBEA to prune the recursion tree, the total number of subsets examined by MBEA is much less than 2^n. Since the least number of nodes on a recursion tree is the total number of all maximal bicliques, we analyze the time complexity in terms of both the number of vertices and the number of maximal bicliques existing in a bipartite graph, as previous works [24, 6] have done.

Lemma 1 All intermediate nodes on the recursion tree represent maximal bicliques.

Proof. Each tree node represents either a maximal or non-maximal biclique. In the instance of a non-maximal node, there are only two reasons it can be made not maximal: either a candidate or a former candidate is connected to all vertices in its L. In the first case, such a candidate will be moved to its R right away according to Observation 6, and the path is shortened. Figure 4.2 is an example of this case, where on the left path of recursion tree T_4, vertex v_2 is directly added to R because it is adjacent to both u_1 and u_3. In the second case, such a former candidate leads to no more maximal biclique from that branch, thus becoming the end of the branch. In other words, non-maximal nodes with a former candidate connected to all vertices in its L are all leaf nodes. Therefore, all intermediate nodes on the recursion tree are maximal. ■

Theorem 3 Given a bipartite graph $G = (U \cup V, E)$ where $|U| = m, |V| = n, m \geq n$, the worst-time complexity of Maximal Biclique Enumeration Algorithm finding all maximal bicliques in G is $O(dn^2\mathcal{B})$, where d is the maximum degree of vertices in V and $\mathcal{B}$ is the number of maximal bicliques.



(a) Recursion tree T_3 (b) Recursion tree T_4

Figure 4.2: Recursion trees without (a) and with (b) Observation 6 on a bipartite graph. The path in dashed lines with three nodes in recursion tree T_3 is compressed to two nodes in recursion tree T_4 by the direct addition of vertex v_2 to R and subsequent elimination of the intermediate non-maximal biclique.

Proof. As proven in Lemma 1, MBEA expands only the nodes that are maximal bicliques on the recursion tree, which means it creates only maximal bicliques as intermediate nodes on the tree and non-maximal bicliques can only be leaf nodes. In other words, the number of non-maximal bicliques created on the tree is at most the total number of the leaf nodes. For any intermediate node on the recursion tree, the number of its children that are leaf nodes representing non-maximal bicliques is less than $n-1$. Thus, the total number of nodes on the recursion tree is at most equal to n times the number of intermediate nodes, which is $O(n\mathcal{B})$ in worst case. Therefore, combining with the time complexity $O(dn)$ at each node, the time complexity of MBEA is $O(dn^2\mathcal{B})$. ■

To understand algorithmic complexity in more detail, MBEA was analyzed using the concept of *delay time*, which is defined as the running time between the output of two consecutive maximal bicliques following the definition given in [40]. MBEA is a polynomial delay time algorithm because the time elapsed between the output of the (k-1)-st and k-th maximal biclique is polynomial bounded in n and d.

Theorem 4 Maximal Biclique Enumeration Algorithm is a polynomial delay time algorithm that outputs maximal bicliques in $O(dn^2)$ time.

Proof. As proven in Theorem 3, MBEA takes $O(dn)$ time to explore one node on the recursion tree, and it explores at most $n-1$ nodes before finding and outputting a maximal biclique. Thus, the worst delay time complexity of MBEA is $O(dn^2)$, and this proves MBEA is a polynomial delay time algorithm. ■

4.2 Improvements to MBEA Algorithm

The MBEA algorithm is improved by two methods: (a) a method to remove vertices from the candidate set earlier; and (b) a method to select vertices from the candidate set in order of their degree. Both methods further prune the recursion tree by avoiding the generation of some non-maximal nodes.

4.2.1 A tree pruning method

The first improvement that further prunes the recursion tree is an extension of Observation 6, which is, if P contains a subset S of vertices that are adjacent to all vertices in L, then $(L, R \cup S)$ is a maximal biclique. Observation 6 allows the direct addition to the biclique of the candidates that have a neighborhood containing the neighborhood of selected candidate x. However, upon the return of the recursive function, MBEA treats the vertices in S the same as other vertices in the candidate set. That is, every vertex in S is still selected to form a branch, some of which lead to non-maximal nodes only. The generation of such branches can be avoided if vertices in S are then grouped to two subsets based on their neighborhood. For a vertex $v \in S$, (a) the neighborhood of v is a proper superset of the neighborhood of the selected candidate x (i.e. $N_L(v) \supset N_L(x)$), that is, x is locally dominated by v; or (b) its neighborhood is exactly same as that of x (i.e. $N_L(v) = N_L(x)$), that is, x locally dominates v. Then, vertices of the second group can be directly moved to the former candidate set Q upon the return of the recursive function, because any biclique that excludes x but includes v is a subgraph of a biclique including both x and v since v is dominated by x. This observation is formulated as follows:

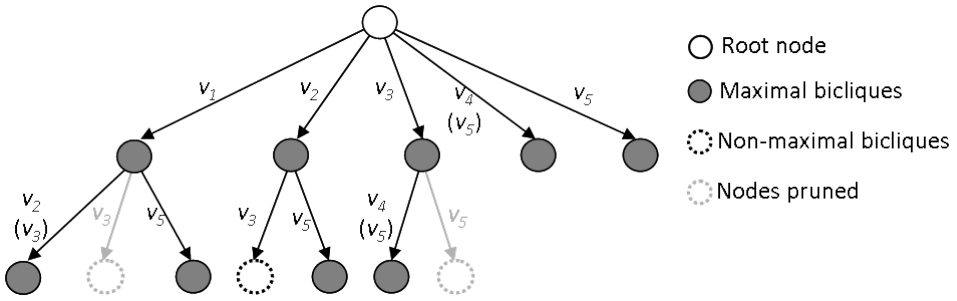


Figure 4.3: A recursion tree with Observation 8 on the bipartite graph in Figure 4.1 (a).

Observation 8 Let x be a selected candidate from P, let $N_L(x)$ denote $\{u \mid u \in L,\ (u,x) \in E\}$, let $C(x)$ denote $\{v \mid v \in P,\ N_L(v) = N_L(x)\}$, and let P_{xv} denote $P \setminus \{x, v\}$, for $v \in C(x)$. For any subset L' of L, and any subset P' of P_{xv}, $(L', R \cup \{v\} \cup P')$ is not a maximal biclique.

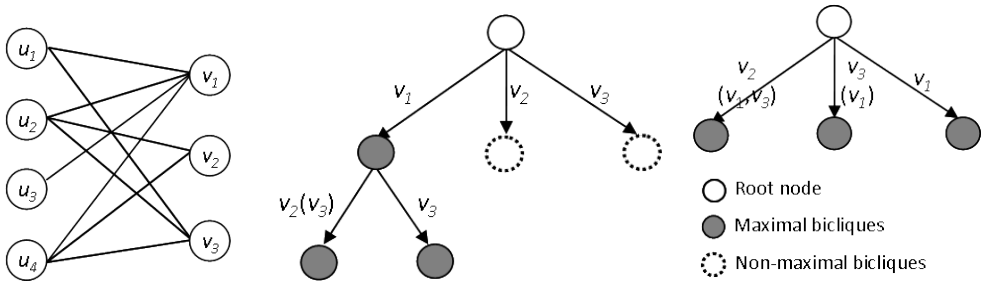
Using the recursion tree T_2 in Figure 4.1 (b) as an example, the branch $\{v_1, v_3\}$ can be pruned since vertex v_3 has the same neighborhood $\{u_1, u_2\}$ as vertex v_2 in the set $L' = N_L(v_1) = \{u_1, u_2, u_6\}$. The biclique involving v_3 has been generated on the branch $\{v_1, v_2, v_3\}$ where v_3 is directly added to the biclique when v_2 is selected. On the other hand, branch $\{v_5\}$ must be kept even though v_5 is added directly into a biclique on the branch $\{v_4\}$ because v_4 is dominated by v_5 in the original L (i.e. $N_L(v_5) \supset N_L(v_4)$). In this case two non-maximal nodes on T_2 can be further pruned to form a new recursion tree of size 10 excluding the root, as shown in Figure 4.3.

4.2.2 A candidate selection method

The MBEA algorithm chooses candidates in canonical order, which is equivalent to random selection. The second improvement was inspired by the observation that the branches explored earlier (left branches) on the recursion tree have more candidates to generate sub-branches than those searched later (right branches) if the selected candidates have the same number of connections to L. One case in point is a connected bipartite graph $G = (U \cup V, E)$ where $|U| = 4, |V| = 3$ and vertex $v_1 \in V$ is adjacent to all vertices in U (Figure 4.4 (a)). If v_1 is the first candidate selected (Figure 4.4 (b)) as MBEA does, then both v_2 and v_3 are candidates for the same level because both connect to at least one vertex in U. However, both nodes $\{v_2\}$ and $\{v_3\}$ are non-maximal since they are subgraphs of bicliques including v_1. On the other hand, if v_1 is the last selected candidate (Figure 4.4 (c)), then there is no vertex left in the candidate set because v_2, v_3 have been explored earlier. Furthermore, v_1 is directly added to all bicliques according to Observation 6 since v_1 is adjacent to all vertices in L. Therefore, the selection of candidates by their degrees in increasing order avoids the generation of some non-maximal nodes. Moreover, it leads to more balanced recursion trees, which is a useful feature for load-balanced parallelization.

4.2.3 Algorithmic Details

The core recursive function `biclique_find_v2()` of the improved algorithm is given in Algorithm 5, where the changes from Algorithm 4 are preceded by a star (*). To distinguish the two MBEA algorithms, this version with improvements is referred to as *Improved MBEA*, or iMBEA. An efficient method is used to reduce the time spent on sorting the candidates by their degrees, since the sorting procedure takes at least $O(k \log k)$ time (k is the size of candidate set) and is applied every time the recursive function `biclique_find_v2()` is called, which can easily become a bottleneck.



(a) (b) (c)

Figure 4.4: Recursion trees without (b) and with (c) the candidate selection method on a bipartite graph (a). Two non-maximal nodes are pruned by selecting candidates in increasing degree order.

An online insertion sorting algorithm is thus used when a vertex is added to the new candidate set. There is a clear tradeoff between the time saved by reduced search space and the time spent on sorting the candidates. If the given bipartite graph has vertices with regular degree, this method worsens the performance rather than improves. The candidate selection method performs best when a graph has diverse degrees of vertices in the smaller vertex set.

An Example

Figure 4.5 is an example of the recursion trees of MBEA before and after improvements for a bipartite graph $G = (U \cup V, E)$, with $|U| = 8, |V| = 5, |E| = 20$ (Figure 4.5 (a)). Vertices $v_i \in V, i \in [1..5]$ have degrees in decreasing order $d(v_1, v_2, v_3, v_4, v_5) = (6, 5, 4, 3, 2)$. The original MBEA algorithm selects vertices from the candidate set P in their canonical order: v_1, v_2, v_3, v_4, v_5 with the number of candidates $(4, 3, 1, 0, 0)$ respectively. Its recursion tree T_5 (Figure 4.5 (b)) has sixteen nodes, excluding the root, of which four are non-maximal and twelve are maximal ones. iMBEA selects the vertices from the candidate set P in non-decreasing order of their degrees: $v_5(2, 2),\ v_4(3, 3),\ v_3(4, 2),\ v_2(5, 1),\ v_1(6, 0)$, with the vertex degree and the candidate number listed in the parentheses following each vertex. The recursion tree T_6 of the improved algorithm (Figure 4.5 (c)) has only twelve nodes, excluding the root, all of which are maximal. Vertex v_1 is selected first and v_5 last in T_5 leading to the generation of two non-maximal nodes $\{v_1, v_5\}$ and $\{v_5\}$, while v_1 is selected last and v_5 first in T_6, which avoids the generation of these two non-maximal nodes.

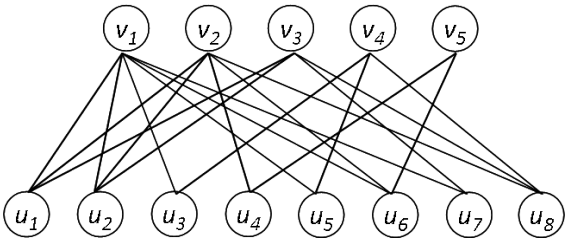
4.3 Performance Evaluation

MBEA is the first general-purpose algorithm designed to enumerate all bicliques in bipartite graphs. Previous approaches either set restrictions on the problem, or had no restrictions but were designed for general graphs or transformed to other problems. MBEA should be much faster than even the fastest known algorithm for general graphs. In order to verify whether this is indeed the case, MBEA was implemented and compared to two algorithms: MICA [6], the best algorithm for finding bicliques on general graph with no restrictions, and FPclose [34, 86], a state-of-the-art algorithm for mining closed patterns. An implementation of MICA on bipartite graphs is available at `http://genome.cs.iastate.edu/supertree/download/biclique/README.html`. Implementations of FPclose are available at `http://fimi.cs.helsinki.fi/src/`, and an improved implementation with less memory consumption [86] was used for our comparison.

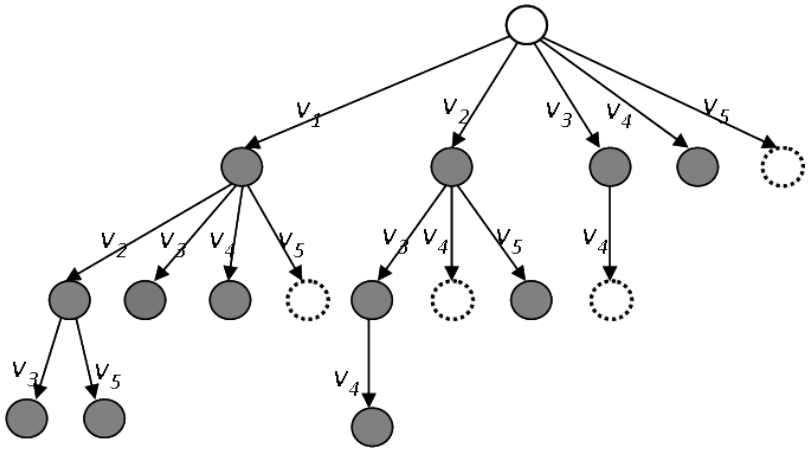
Algorithm 5: The Improved Maximal Biclique Enumeration Algorithm (*Improved MBEA*)

procedure `biclique_find_v2`(G, L, R, P, Q);
G: a bipartite graph $G = (U \cup V, E)$;
L: set of vertices $\in U$ that are common neighbors of vertices in R;
R: set of vertices $\in V$ belonging to the current biclique;
P: set of vertices $\in V$ that can be added to R;
Q: set of vertices $\in V$ that have previously been added to R;

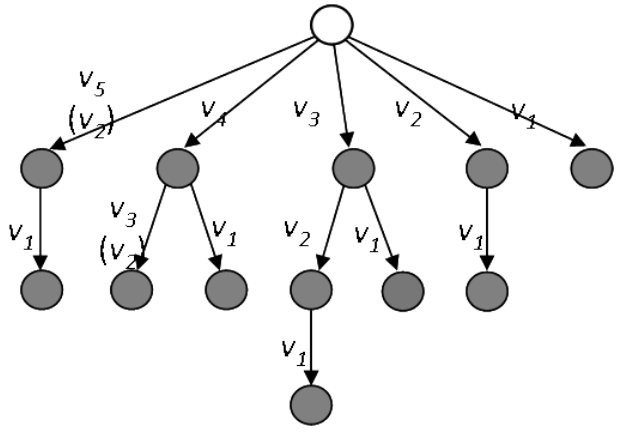
* $i \leftarrow 0$; `// Position of selected candidate in` P
while $P \neq \emptyset$ **do**
  * $x \leftarrow P[i++]$; `// Select next candidate from` P `in order`
  $R' \leftarrow R \cup \{x\}$;
  $L' \leftarrow \{u \in L \mid (u, x) \in E(G)\}$; `// Observation 4: extend biclique`
  *$\overline{L'} \leftarrow L \setminus L'$; $C \leftarrow \{x\}$;
  $P' \leftarrow \emptyset$; $Q' \leftarrow \emptyset$; `// Create new sets`
  `// Observation 5: check maximality`
  $is\_maximal \leftarrow TRUE$;
  forall v *in* Q **do**
    $N[v] \leftarrow \{u \in L' \mid (u, v) \in E(G)\}$;
    `// Observation 7: end of branch`
    if $|N[v]| = |L'|$ **then**
      $is\_maximal \leftarrow FALSE$;
      break;
    else if $|N[v]| > 0$ **then** $Q' \leftarrow Q' \cup \{v\}$;
  if $is\_maximal = TRUE$ **then**
    forall v *in* P, $v \neq x$ **do**
      $N[v] \leftarrow \{u \in L' \mid (u, v) \in E(G)\}$;
      if $|N[v]| = |L'|$ **then**
        $R' \leftarrow R' \cup \{v\}$; `// Observation 6: expand to maximal`
        * $S \leftarrow \{u \in \overline{L'} \mid (u, v) \in E(G)\}$;
        * **if** $|S| = 0$ **then** $C \leftarrow C \cup \{v\}$; `// Observation 8: further pruning`
      else if $|N[v]| > 0$ **then**
        * `INSERT_INC(`P', v, N`);` `// Insert` v `to` P' `in non-decreasing order of` N
    `PRINT(`L', R'`);` `// Report maximal biclique`
    if $P' \neq \emptyset$ **then** `biclique_find_v2(`G, L', R', P', Q'`);`
  `// Move` C `from candidate set to former candidate set`
  * $Q \leftarrow Q \cup C$; $P \leftarrow P \setminus C$;



(a) A bipartite graph G;



(b) A recursion tree T_5 of MBEA;



(c) A recursion tree T_6 of iMBEA.

Figure 4.5: The recursion tree of the improved MBEA for a bipartite graph G with $|U| = 8, |V| = 5, |E| = 20$ (a) compared to original algorithm. There are 12 maximal bicliques existing in G. The recursion tree T_5 of original algorithm (b) searches a total number of 16 nodes excluding the root with 75% maximal nodes, while the tree T_6 of improved algorithm (c) searches only 12 maximal nodes. The search space of this case is reduced by about 25% by the improvement.

To understand the performance of the MBEA algorithms, experiments were performed on a Dell GX280 with dual 3.4GHz Pentium 4 processors and 2.0GB SDRAM, running Linux 2.6.8-2-686-smp kernel. Since the implementations of MBEA, MICA and FPclose are all sequential, they were begun on a single processor in all test runs. The MBEA algorithms were implemented in `C` and the codes were compiled with the GNU `gcc` with the O3 optimization flag turned on. Both MICA and FPclose implementations were also compiled with O3 turned on.

In all experiments, the average wallclock run times that included both the I/O times and the computation times for finding all the maximal bicliques, and the average delay times for outputting each maximal biclique are reported. These runtimes were obtained as the average of ten runs, five runs, or three runs for graphs that can be finished within a minute, an hour, or three days, respectively. Programs were killed after three days running and runtimes of those cases are not reported.

For both MBEA and MICA, the input is an edge list of a bipartite graph and the output is a list of maximal bicliques with both vertex sets. For FPclose, the input is an adjacency list for the smaller vertex set of a bipartite graph, and the output is a list of maximal bicliques with vertices on only one side. The run times of FPclose do not include the time to convert inputs and outputs to proper formats.

4.3.1 Results on Biological Graphs

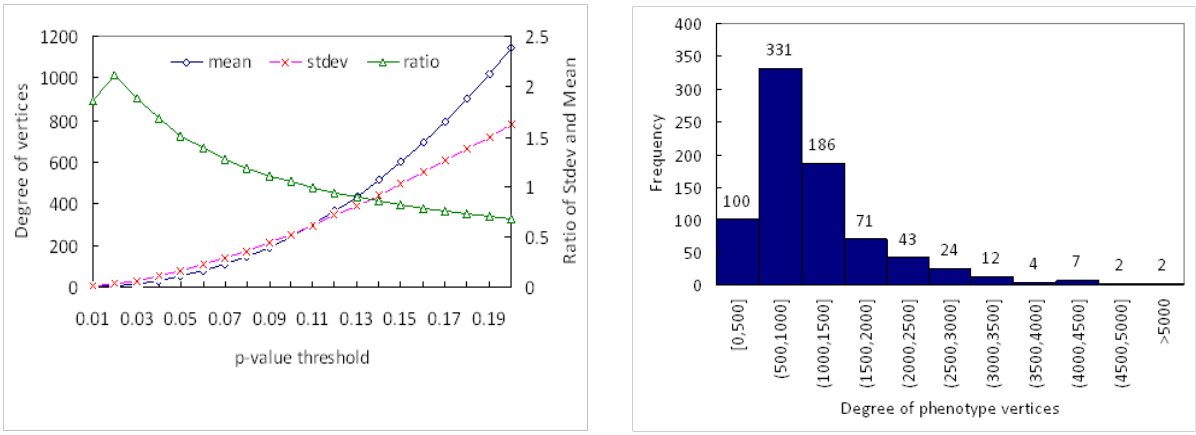
Focusing on ontological discovery [19], we firstly applied MBEA, MICA and FPclose to mouse gene expression data with 45,137 probesets (genes) and 782 phenotypes. A p-value matrix was first calculated from the gene expression data and the phenotype data using t-statistics. The p-value matrix (weighted bipartite graph) was then transformed to binary matrices (unweighted bipartite graph) by using various thresholds from 0.01 to 0.20 with a step of 0.01, which means that those edges with a weight less than or equal to the given threshold were kept.

The resulting bipartite graphs vary in both order and edge density. Graph profiles including the number of genes, phenotypes, and edge densities at various thresholds, are shown in Table 4.1. Both order and density increase at low thresholds until the number of phenotypes (i.e. the size of smaller vertex set m) reaches the maximum number (782) and stops growing at threshold 0.05. The number of genes grows at a slower speed and reaches its maximum (45,137) at a very high end (0.19). At all thresholds, edge density keeps increasing from 0.2% to about 2.5%. Therefore, the problem complexity increases quickly with the increasing of p-value thresholds due to the growing graph in both order and density. Furthermore, the number of maximal bicliques in bipartite graphs grows exponentially with the linearly increasing of p-value thresholds.

To understand the degree structure of these biological graphs, the means and standard deviations of the phenotype degrees at various thresholds are plotted in Figure 4.6 (a). These biological graphs show a large variance in phenotype degrees. The ratios show that standard deviation gradually decreases from twice the mean degree at threshold 0.02 to about 60% of the mean at the highest threshold, except the small jump at the lowest threshold. Figure 4.6 (b) shows the distribution of phenotype degrees in the bipartite graph generated at threshold 0.20. The degrees roughly follow a Pareto distribution with almost half of the phenotypes ranging within [500,1000]. The minimum and maximum degrees are 175 and 5,691 respectively. The diversity of phenotype degrees leads to better performance of the *Improved MBEA* than MBEA as is shown later in this section.

Table 4.1: Profile of the bipartite graphs generated from mouse gene expression data.

p-value threshold	0.01	0.02	0.03	0.04	0.05	0.06	0.07
# Gene (n)	1104	4232	9316	15457	21977	28212	33390
# Phenotype (m)	397	679	760	780	782	782	782
Edge density (e%)	0.31	0.20	0.19	0.21	0.24	0.28	0.33
# Biclique ($\mathcal{B}$)	454	1264	2770	6193	12745	24678	45466
p-value threshold	0.08	0.09	0.10	0.11	0.12	0.13	0.14
# Gene (n)	37382	40427	42477	43712	44437	44819	45002
# Phenotype (m)	782	782	782	782	782	782	782
Edge density (e%)	0.40	0.48	0.57	0.69	0.82	0.97	1.14
# Biclique ($\mathcal{B}$)	81673	143957	250474	433568	775004	1352008	2390457
p-value threshold	0.15	0.16	0.17	0.18	0.19	0.20	
# Gene (n)	45088	45119	45130	45133	45137	45137	
# Phenotype (m)	782	782	782	782	782	782	
Edge density (e%)	1.33	1.53	1.76	2.00	2.26	2.54	
# Biclique ($\mathcal{B}$)	4283178	7486034	13101882	22552313	39944520	70702688	



(a) (b)

Figure 4.6: The mean and standard deviation of degrees of phenotype vertices in bipartite graphs generated at various p-value thresholds (a), and the distribution of the phenotype degrees in the graph generated at threshold 0.20 (b).

Comparison of MBEA with MICA and FPclose

Figure 4.7 reports the wallclock run times and delay times of MBEA compared with MICA and FPclose on those biological bipartite graphs. Figure 4.7 (a) shows that MBEA, MICA and FPclose perform closely on the small and sparse graphs, and that MBEA well outperforms both MICA and FPclose with an increasing number of maximal bicliques, although FPclose performs best on the smallest graph. Both MBEA and FPclose are faster than MICA because they are targeted at bipartite graphs. However, FPclose runs out of memory from threshold 0.09, which makes it impractical for solving graphs generated from real biological data sets. It is not difficult to see from the log-scaled run times (Figure 4.7 (b)) that MBEA is about three orders of magnitude faster than MICA starting from p-value threshold 0.07. MBEA can finish within a second, FPclose takes a few seconds, and MICA needs a minute for those small and sparse graphs. For graphs that MBEA can solve within a minute, MICA may need several hours while FPclose runs out of memory. For those larger and denser graphs, MBEA finishes within an hour, or in some cases several hours, while MICA cannot finish after running three days (thus run times are not available for MICA on those graphs). Figure 4.7 (c,d) shows that MBEA is nearly a hundred times faster than MICA in terms of delay time (the average time to output a maximal biclique). The average delay time of MBEA stays at around 0.1 milliseconds, while those of MICA increase quickly to more than fifty milliseconds. At threshold 0.08, the delay times of MBEA, FPclose and MICA are 0.03, 5.7, 14.4 milliseconds, respectively.

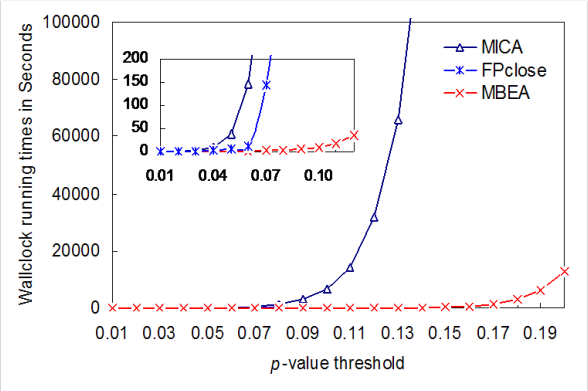
To understand the reasons for the performance difference between MBEA and MICA, both implementations were traced in order to calculate the total number of times that each fundamental operation is executed and the cumulative runtimes of each operation. For both algorithm implementations, there are three fundamental operations, (a) the I/O operation to output a maximal biclique, (b) the EXPAND operation to expand the search space, and (c) the CHECK operation to check whether a biclique is maximal.

Figure 4.8 shows that the EXPAND operation takes about half of the total execution time in both implementations. MBEA spends 20% to 30% of the time on checking the maximality of bicliques found at each recursion tree node, compared to MICA at 40%, due to its simple maximality check method. Consequently, the time percentage of the I/O operation in MBEA is larger than that in MICA. Furthermore, in both implementations, the I/O time percentage gradually decreases with an increased number of maximal bicliques. This is because the total execution time is quickly increased with the number of maximal bicliques but the I/O time is linearly increased. For example, when the maximal biclique number increases to 45,466 with p-value 0.07, the I/O time percentage of MICA decreases to only 0.8%.

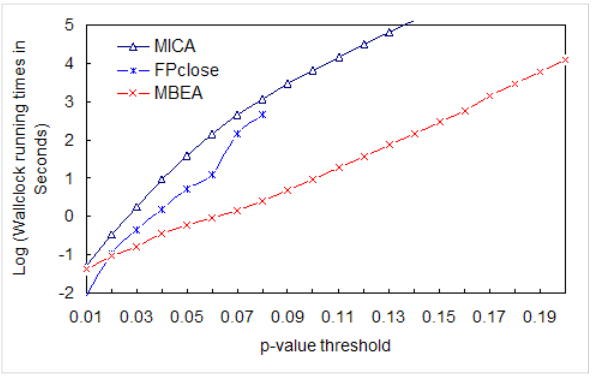
Nevertheless, the main reason that MBEA greatly outperforms MICA is that their search space sizes are so different. As shown in Figure 4.9 (a), the search space represented as the total number of times that are performed for finding all maximal bicliques by MICA increases much more rapidly than does that of MBEA as the graph size grows. The ratio of the times and the number of maximal bicliques was also computed (Figure 4.9 (b)) to evaluate the search space, where the ratio by MICA explodes to more than 1,000 at threshold 0.04 while that of MBEA increases linearly as p-value grows and does not exceed 10 until threshold 0.16. The reduced search space in MBEA is achieved by exploiting the structure of bipartite graphs and employing the branch-and-bound technique.

Comparison of MBEA and Improved MBEA

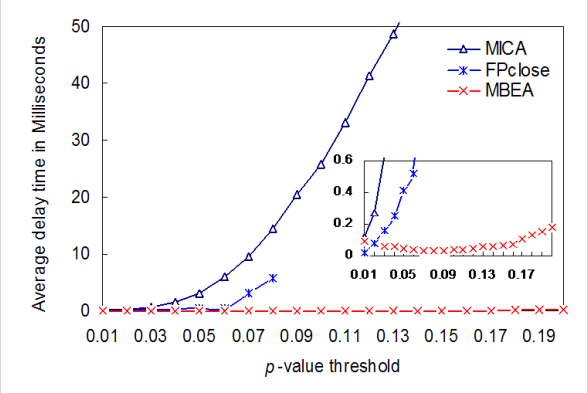
Both MBEA algorithms were applied on the biological bipartite graphs generated from the mouse gene expression data to compare the performance before and after the improvements. The run



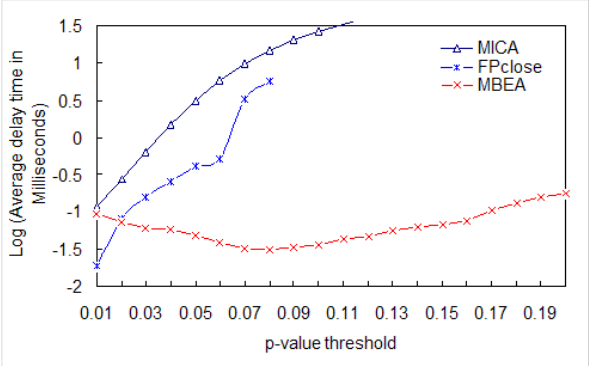
(a) run times



(b) run times in log-scale



(c) delay times



(d) delay times in log-scale

Figure 4.7: Wallclock running times (a,b) and average delay times (c,d) of the algorithm MBEA compared with MICA and FPclose for finding all maximal bicliques on graphs from mouse gene expression data.

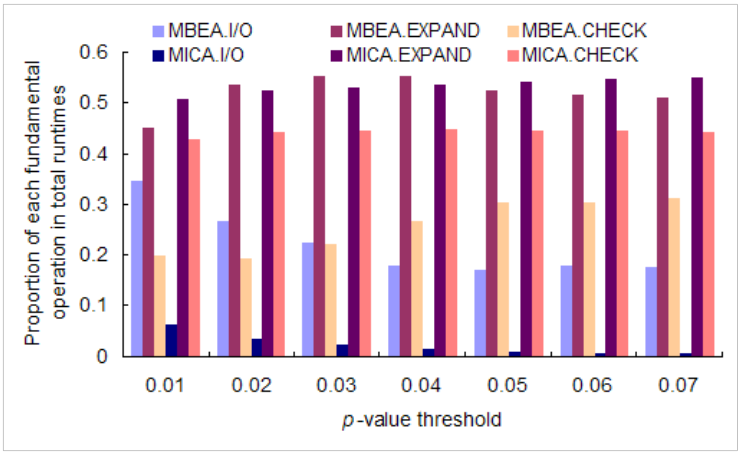
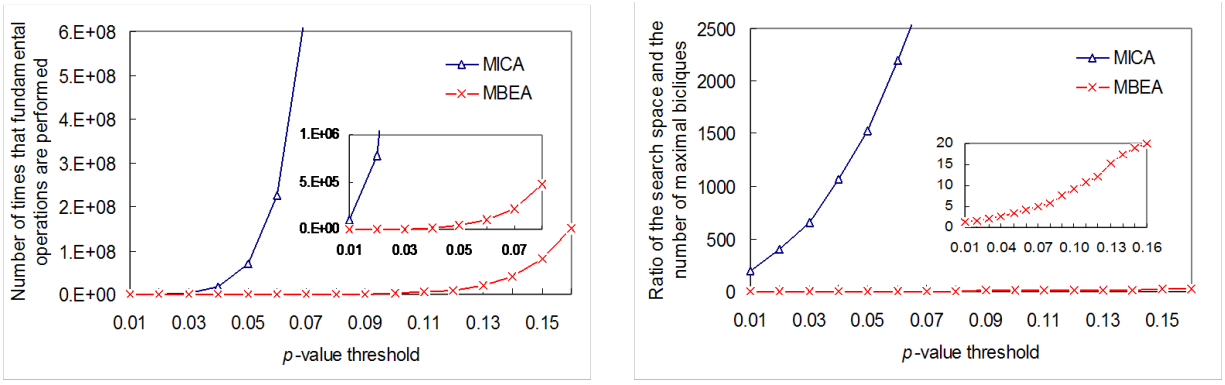


Figure 4.8: Comparison of time distribution by MBEA and MICA on three fundamental operations, including Disk I/O, EXPAND for traversing the search space, and CHECK for checking the maximality of bicliques.



(a) (b)

Figure 4.9: Comparison of search spaces used by MBEA and MICA: (a) the total number of times that fundamental operations are performed for finding all maximal bicliques; (b) the ratio of the nodes in the search space and the number of maximal bicliques, which is the average number of search space needed to be explored for finding one maximal biclique.

times were evaluated, as was search space in terms of the ratio of recursion tree size and maximal biclique number. The run times (Figure 4.10 (a)) of both MBEA algorithms increase gradually as the p-value grows. The two curves crossing at p-value thresholds 0.08 means iMBEA takes longer to finish than does MBEA at low thresholds, but outperforms MBEA at high thresholds. The improvement reaches two times speed increases at threshold 0.20. This result confirms our anticipation in Section 4.2 that iMBEA performs better on graphs with diverse degrees (Figure 4.6 (a)). iMBEA is only slightly slower on very small and sparse bipartite graphs, which is due to the small search space at low thresholds.

Figure 4.10 (b) shows the search space of both MBEA algorithms represented as the ratio of the number of nodes explored and the number of maximal bicliques. The ratio of iMBEA stays around three at high thresholds (> 0.11), while that of MBEA increases to about thirty (Figure 4.10 (b)). As discussed in Section 4.2, the reduced search space by the improved algorithm is achieved by avoiding the generation of non-maximal leaf nodes in the recursion tree.

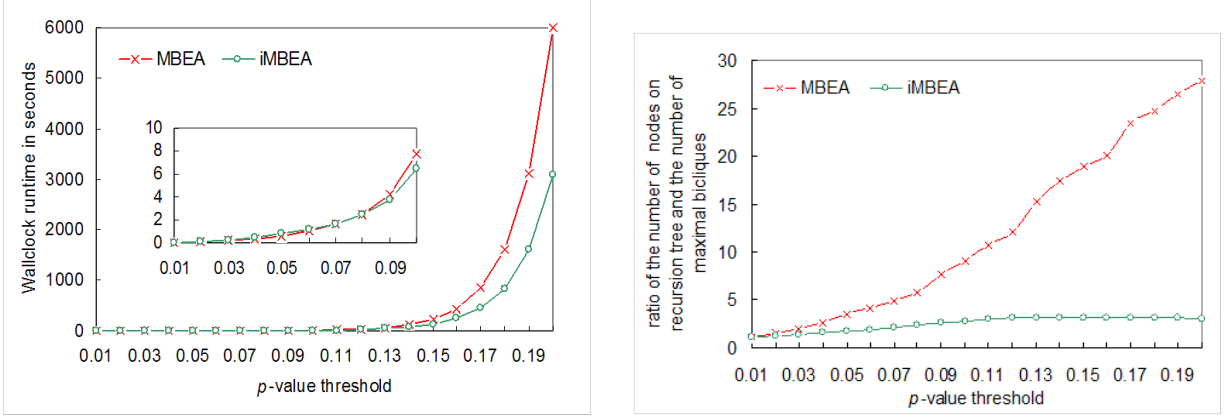
4.3.2 Results on Synthetic Graphs

The MBEA algorithms were also evaluated on random bipartite graphs. A random bipartite graph generator was implemented for this purpose with four parameters:

- m: the size of the larger vertex set,
- n: the size of the smaller vertex set,
- μ: the mean degree of vertices in the smaller vertex set, and
- σ: the standard deviation of degrees of vertices in the smaller vertex set;

and two distributions:

- a Gaussian distribution of degrees of vertices in the smaller set $\varphi_{\mu,\sigma}(d(v)) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where $d(v)$ is degree of a vertex $v \in V$, and



(a) (b)

Figure 4.10: Performance comparison of maximal biclique enumeration by MBEA and iMBEA, including the average wallclock run times in seconds (a) and the ratio of the number of nodes on recursion tree and the number of maximal bicliques (b).

- a uniform distribution for generating edges for each vertex in the smaller set, such that every vertex in the larger set has the equal probability to be connected to the given vertex.

Thus, the size of a bipartite graph is decided by m and n; its edge density e is determined by the average degree μ of vertices in the smaller set and the number of vertices in the larger set m (i.e. $e = \mu/m$); and its degree structure is determined by the Gaussian distribution of μ and the standard deviation σ.

The random bipartite graph generator generates a bipartite graph as follows: Given the smaller vertex set size $|V| = n$, for every vertex v in V, it generates edges that have v as one of the endpoints in two steps: (a) firstly decides the degree of v using the Gaussian distribution with the given mean μ and standard deviation σ, and then (b) for every vertex u in the larger vertex set U, decides whether there is an edge between u and v using the uniform distribution with the edge probability p computed from its degree $d(v)$ over the size of larger vertex set m (i.e. $p = d(v)/m$).

Comparison of MBEAs with MICA and FPclose

Random bipartite graphs with various sizes and edge densities were generated to compare the performance of MBEAs with MICA and FPclose. To test the effect of different parameters on the performance, graphs were generated in three cases:

1. Fixing the smaller vertex set size at $n = 500$ and the edge density at $e = 2\%$, increasing the larger vertex set size m from 1,000 to 10,000 with a step of 1,000;

2. Fixing the larger vertex set size at $m = 10,000$ and the edge density at $e = 2\%$, increasing the smaller vertex set size n from 100 to 1,000 with a step of 100;

3. Fixing the sizes of both vertex sets $m = 10,000$, $n = 500$, increasing the edge density e from 1% to 10% with a step of 1%.

For all three cases, the standard deviation was chosen to be 60% of the average vertex degree, which roughly follows the property of biological graphs shown in section 4.3.1.

Figure 4.11 reports how the run times and delay times on random bipartite graphs are affected by the size of the larger vertex set m (Figure 4.11 (a)), by the size of the smaller vertex set n (Figure 4.11 (b)), and by the edge density e (Figure 4.11 (c)). All three cases show that MBEAs constantly outperform both FPclose and MICA, while FPclose is faster than MICA in all cases except for denser graphs in the third case, for which FPclose ran out of memory. This is consistent with their performances on biological graphs. MBEA and iMBEA perform very similarly on all cases. For both MBEA algorithms, the run time increases slowly as the large vertex set grows, doubles each time the smaller vertex set is enlarged by 100, and also increases exponentially with the increasing of the edge density. Thus, the smaller vertex set size and the edge density has greater effect on the run times of MBEA algorithm than the larger vertex set size.

Comparison of MBEA and Improved MBEA

The performance of MBEA and *Improved MBEA* were further evaluated on random graphs of the same size and edge density but with different degree distributions in order to test the effect of degree structure on the iMBEA. Fixing the bipartite graph size and edge density at $m = 10,000,\ n = 1,000,\ e = 5\%$, the standard deviation of degrees were increased from 0% to 100% of the mean degree with a step of 5%. Figure 4.12 plots the actual average degrees, standard deviations and their ratios, and the average delay times of MBEA and iMBEA. The two curves of delay times crossing in the middle show that iMBEA is faster on graphs with higher degree standard deviations and the delay times of iMBEA decrease as the standard deviation increases. This confirms the expectation that iMBEA performs better on graphs with diverse degrees.

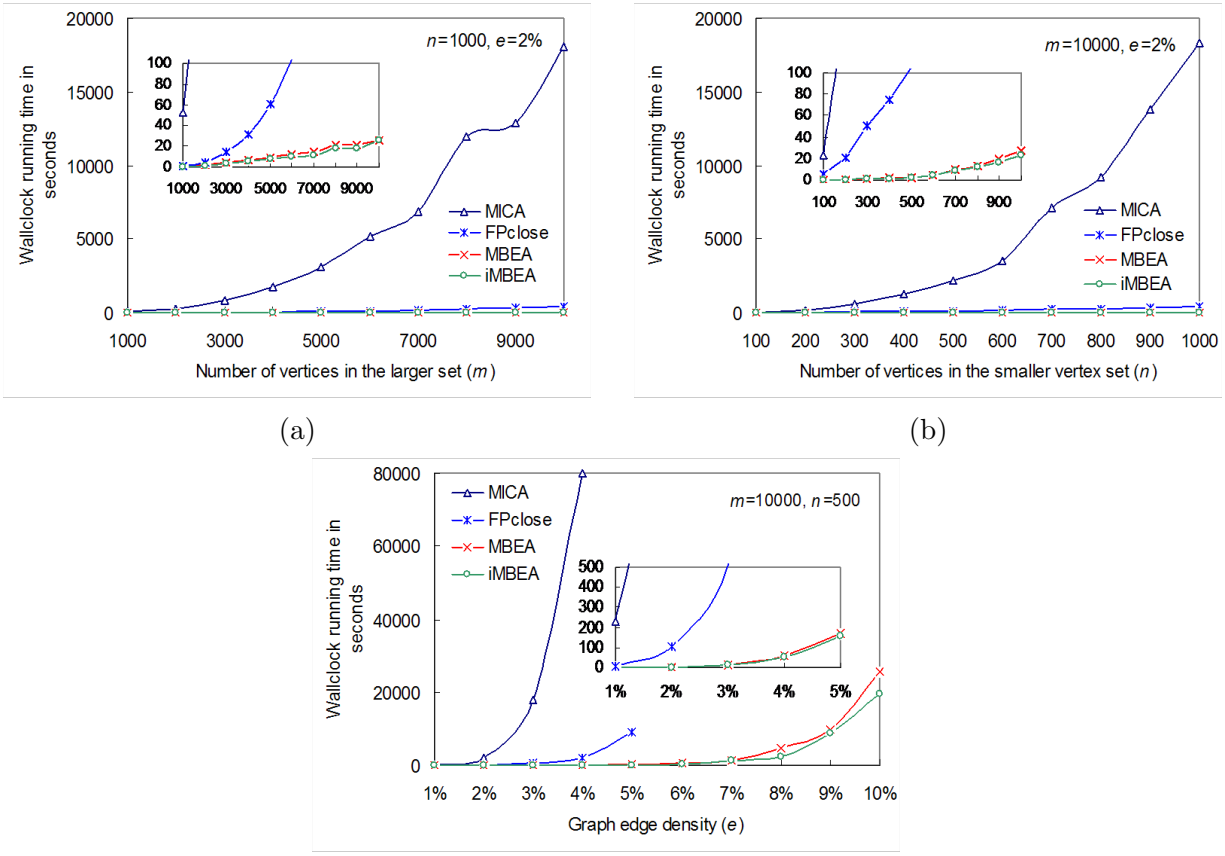


Figure 4.11: Average run times of the MBEA algorithms, MICA and FPclose on random bipartite graphs with various sizes (a,b) and edge density (c).

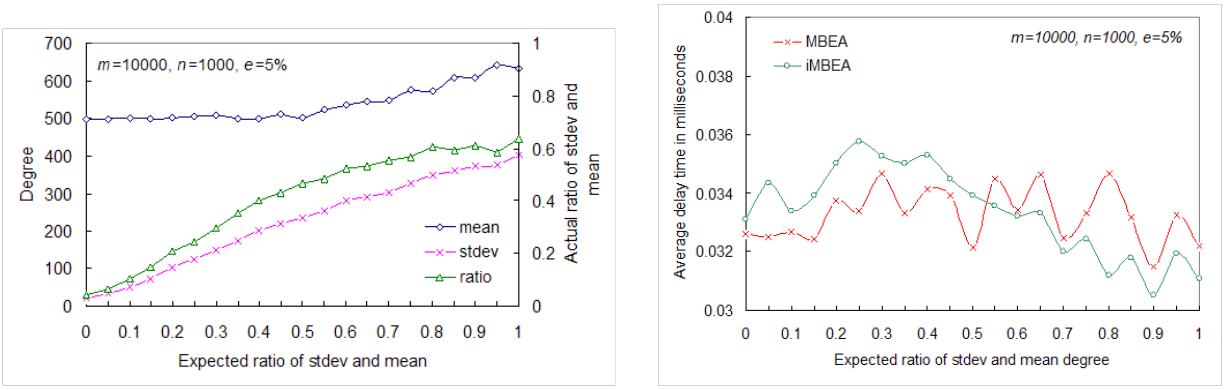


Figure 4.12: Degree structures of random graphs with same size and edge density (a), and average delay times of MBEA and iMBEA on those graphs (b).

Chapter 5

Applications in Genetics

In this chapter, the clique and biclique enumeration algorithms are applied to two applications in genetics: the linkage disequilibrium (LD) analysis in population genetics with MCLEARS, and the ontology discovery for systems genetics with MBEA. The LD study analyzes large sets of genetic variations throughout the genome, and the ontology study integrates multiple genome-scale data sets of heterogeneous types such as gene expression data and phenotype data.

5.1 Application I: Linkage Disequilibrium Analysis

In LD analysis, two genetic loci are said to be associated in LD when some of their allele combinations occur more often than expected assuming the loci segregate completely independently. The identification of LD networks that are groups of associated loci is crucial to characterizing the biological drivers that contribute to LD. Using the clique-based model and the MCLEARS algorithm, LD networks can be comprehensively detected in the graphs constructed from large sets of genetic markers. This method allows the extent of LD at varying linkage thresholds to be evaluated. Analyses of these LD networks provide a quantitative comparison of population architectures.

5.1.1 The Biological Problem

Background

A *DNA sequence*, or genetic sequence, is a succession of letters representing the primary structure of a real or hypothetical DNA strand. There are four possible letters, $\mathcal{A}, \mathcal{G}, \mathcal{C}$, and $\mathcal{T}$, representing the four nucleotide bases of a DNA strand. The size of an individual gene or an organism's entire genome is often measured in *base pairs* because DNA is usually double-stranded. A *locus* (plural *loci*) is a fixed position on a chromosome. A variant of the DNA sequence at a given locus is called an *allele*. Nearly all mammals are *diploid* organisms that have two homologous copies of each chromosome, usually one from the mother and the other from the father; for example, a mouse genome has 20 chromosome pairs. An organism is *homozygous* at a specific locus when it carries the same allele on the two homologous chromosomes or is *heterozygous* when it carries two different alleles.

Genetic *recombination* is the process by which a DNA strand is broken and then joined to a different DNA strand, as shown in Figure 5.1 (a). Recombination commonly occurs during *meiosis* as a chromosomal crossover between the paired chromosomes inherited from each of one's parents. Recombination can occur at any location along a chromosome. Therefore, the frequency of recombination between two locations depends on their distance. Genetic loci that lie near

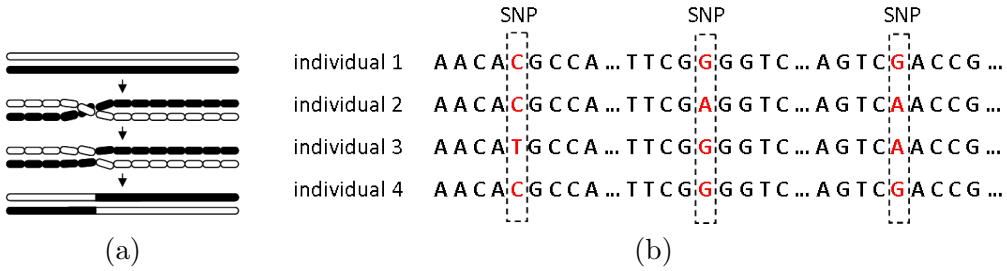


Figure 5.1: Examples of (a) genetic recombination and (b) SNPs.

each other on a chromosome tend to be inherited together because of relatively low recombination frequency, and are thus genetically *linked*. On the other hand, the linkage between distant loci on the same chromosome may be destroyed due to the large degree of crossover. After enough instances of recombination, the correlation between two loci will disappear. Assuming random mating, alleles for loci on different chromosomes are usually not linked due to independent assortment of chromosomes during meiosis. However, if a correlation between any two loci is observed, it is termed *linkage disequilibrium*.

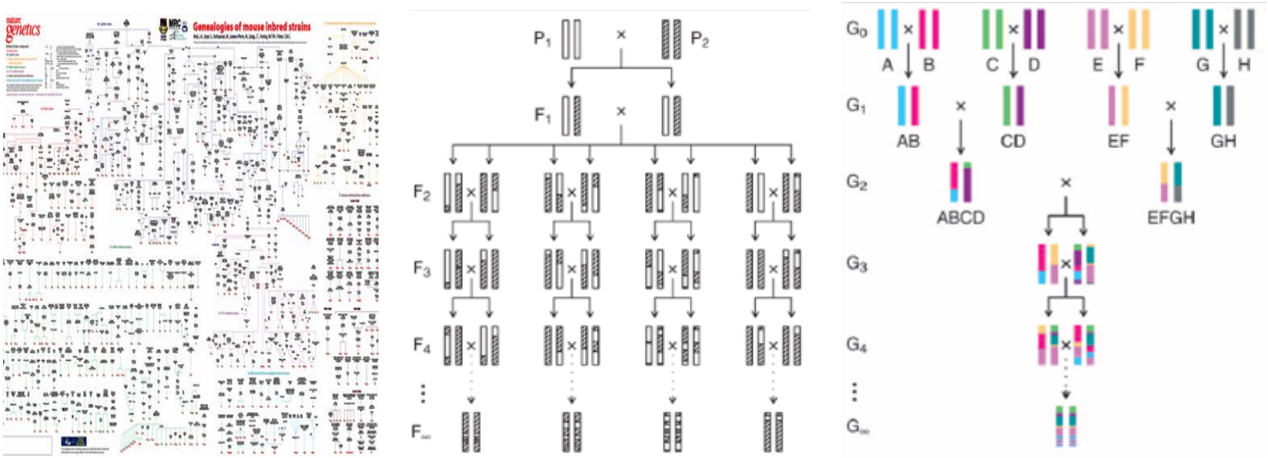
Genetic markers are known DNA sequence variations due to mutation or alteration in the genomic loci. They can be used to study the relationship between an inherited disease and its genetic cause because genetic loci associated with a same trait are usually inherited as a single unit. The simplest form and the most common genetic marker is a short DNA sequence surrounding a *single nucleotide polymorphism* (SNP), which is a single base pair variation in DNA sequence (Figure 5.1 (b)). An SNP usually has only two alleles, where the allele with higher observed frequency in a population is called *major allele* and the other is called *minor allele*. SNPs with minor allele frequency of greater than 0 are *informative SNPs*.

Motivation

The mouse is a prime organism of choice for modeling human disease. Inbred strains of mice have been widely used for this purpose. For example, inbred mice are used in the quantitative trait locus (QTL) analysis to identify genetic loci underlying a quantitative trait of interest such as the weight of a human. However, without knowledge of its genomic structure the suitability of an inbred mouse population for QTL analysis is not clear. When an individual has long-range blocks of linked loci, the precision of its genome becomes low, and it is difficult to identify QTLs accurately. It is critical to understand the LD structure of these populations.

The structure of inbred mouse populations is determined by their breeding history and particularly, generations of out-crossing, random or non-random mating, and progenitor diversity. Unlike standard inbred (SI) strains, the two-progenitor recombinant inbred (RI) sets, such as BXD strains and the eight-progenitor Collaborative Cross (CC) strains, are large systematically inbred populations. These populations vary in size, founder diversity, breeding design, and actual breeding history (Figure 5.2). Standard inbred strains have a longer non-random mating history than other strain sets. Both RI and CC strains are randomly mated, but the CC set originated from eight progenitors and had two more generations of out-crossing than 2-way RI strains.

The hypothesis is that the non-random breeding history of standard inbred strains should result in groups of linked loci located on multiple chromosomes, while most linked loci groups of randomly mated RI lines should be short-range or long-range on the same chromosome. The CC strains should have smaller LD blocks than RI lines because their genomes have been better shuffled with three



(a) (b) (c)

Figure 5.2: The breeding history of standard inbred strains (a), two-way recombinant inbred strains (b), and eight-way Collaborative Cross strains (c).

generations of out-crossing and have fewer groups on multiple chromosomes than SI strains because of random mating. In order to confirm this hypothesis, *LD networks* that contain groups of linked loci need to be identified throughout the entire genome using large sets of genetic markers. The clique-based model was thus developed to detect LD networks using MCLEARS.

Data Sets

The data sets analyzed in this research are SNP sets of multiple inbred mouse populations. They are (a) a set of 67 SI mice consisting of several sub-populations, (b) three sets of two-progenitor RI mice including 89 BXD strains, 77 LXS strains, and 36 AXB and BXA strains, and (c) a set of 38 eight-progenitor CC RI strains. Among these five sets, SI, BXD, LXS, and AXB/BXA strains were genotyped at 13,377 genetic markers and individuals from the CC set were genotyped at 11,970 markers.

Table 5.1 lists the number of strains, the number of informative SNPs, the number of progenitors, and the number of outcrossing generations in each population. The SI set has the largest number of informative SNPs among these five populations. Each SNP set is a two-dimensional matrix with rows representing SNPs, columns representing strains, and values coming from one of the four letters $\mathcal{A}, \mathcal{G}, \mathcal{C}$, and $\mathcal{T}$. A SNP set can be transformed to a binary matrix since each SNP has only two alleles, where the major alleles are represented as 1s and the minor alleles as 0s. Therefore, the data sets can be converted to binary matrices as inputs to LD analysis.

5.1.2 The Clique-Based Model

A graph theoretical approach, specifically a clique-based model, was developed to detect and analyze the linkage disequilibrium networks in a population to evaluate its genetic structure. Using the clique-based model, it is possible to detect LD networks using large SNP sets throughout the entire genome. Figure 5.3 is an overview of the major components in the clique-based graph model: (a) a statistical module for computing the LD coefficient matrix by pairwise LD measures, (b) a mapping module for thresholding and constructing unweighted graphs at each threshold, (c) a computational module for detecting the LD networks using maximal clique algorithms such as

Table 5.1: SNP sets of inbred mouse populations studied in LD analysis

Populations	#Strains	#SNPs	#Progenitors	#Outcrossing
SI	67	12,872	-	-
BXD	89	7,463	C57BL/6J, DBA/2J	1
LXS	77	4,829	ILS, ISS	1
AXB/BXA	36	7,919	A/J, C57BL/6J	1
CC	38	10,691	8 Strains	3

MCLEARS and paraclique, and (d) an analytical module for quantitatively comparing the LD networks. This approach allows the evaluation of the LD extent at varying edge-weight thresholds, which produce unweighted graphs for more in-depth analysis. The number, size, and location of the SNPs contained in LD networks can be analyzed to compare populations and sub-populations quantitatively.

Computation of SNP-SNP LD Coefficient Matrix

First, the statistical module takes as input a SNP set to compute a matrix of linkage disequilibrium coefficients for all SNP pairs. The SNP-SNP LD coefficient matrix can be computed by one of the pairwise LD measures, including Lewontin's D' [48], the correlation coefficient r, and the entropy-based mutual information (MI).

$$D' = \begin{cases} \frac{D}{\min(p_1q_0, p_0q_1)} & \text{if } D > 0 \\ \frac{D}{\min(p_1q_1, p_0q_0)} & \text{if } D < 0 \end{cases} \in [-1, 1], \tag{5.1}$$

$$r = \frac{D}{\sqrt{p_1p_0q_0q_1}} \in [-1, 1], \tag{5.2}$$

$$MI = \sum_{i,j \in \{1,0\}} p_{ij} \log \frac{p_{ij}}{p_i p_j} \in [0, 1], \tag{5.3}$$

where $D = p_{11} - p_1q_1$ is the difference between the observed allele combination frequency in a given population and the expected allele combination frequency assuming independence between two SNPs. The observed ($p_{ij},\ i, j \in \{0, 1\}$) and the expected ($p_iq_j,\ i, j \in \{0, 1\}$) combination frequencies of two SNPs with their allele frequencies are shown in Table 5.2.

There are three differences between Lewontin's D', correlation coefficient r, and mutual information: (a) D' and r are strongly influenced by allele frequencies while entropy-based mutual information is less influenced, (b) D' and r measure both positive and negative LD while mutual information is not sign sensitive, and (c) D' and r work for only two alleles while mutual information is applicable to multiple (> 2) alleles. In this study, the reported results were estimated using mutual information.

The computation of LD matrix was performed on the SNP sets after the removal of non-informative SNPs and the handling of missing data and heterozygous genotypes. When comparing multiple populations, the size of the smallest population was chosen as a standard size for every population. For each population, a bootstrapping procedure was applied to randomly draw samples with the standard size from the population and repeat the procedure for a specific number of times until average coefficient values become stable.

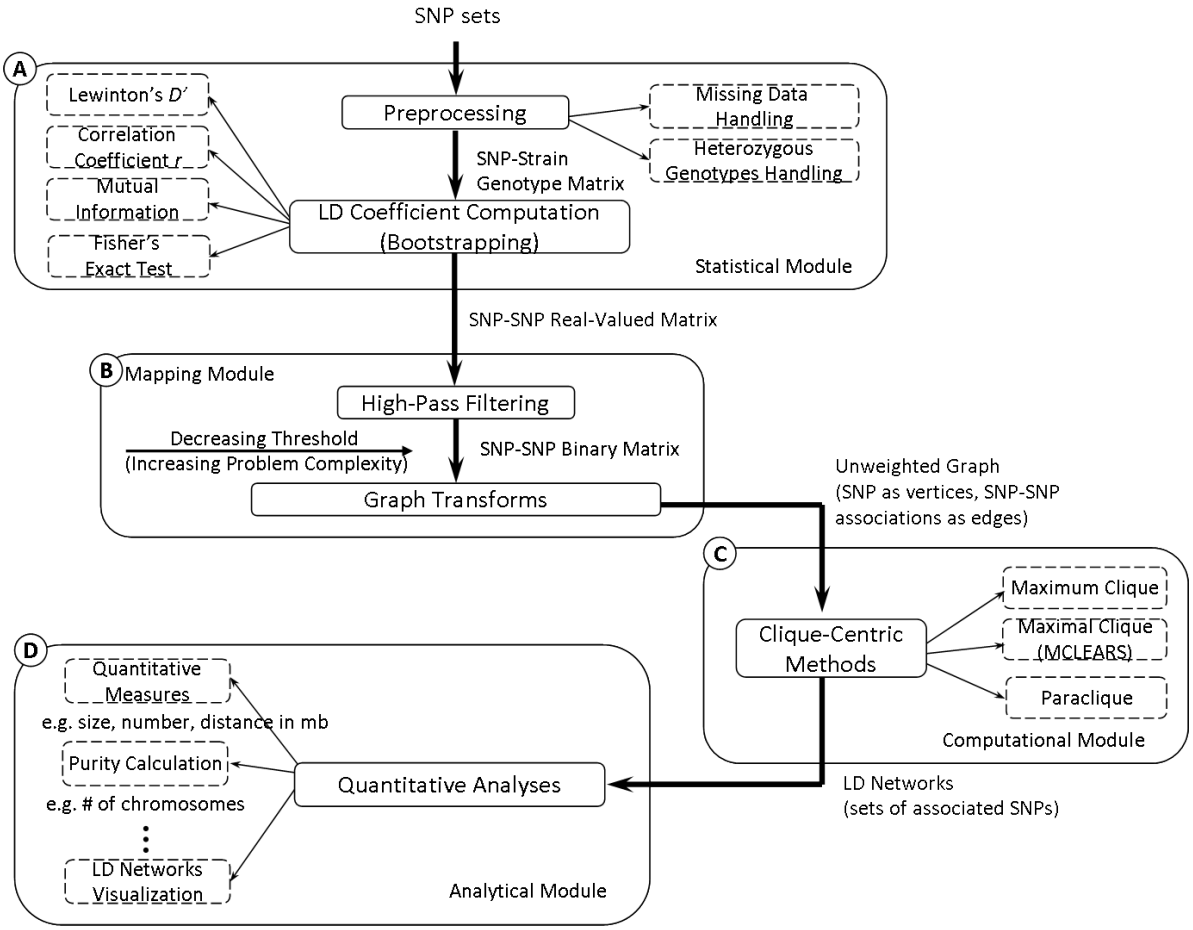


Figure 5.3: An overview of the clique-based model for LD analysis using SNP sets. The model include (A) a statistical module for computing the LD matrix, (B) a mapping module for thresholding and constructing unweighted graphs at each threshold, (C) a computational module for detecting the LD networks using maximal clique algorithms such as MCLEARS and paraclique, and (D) an analytical module for quantitatively comparing the LD networks.

Table 5.2: The observed and the expected combination frequencies between two SNPs

			Alleles at SNP Y: 1	Alleles at SNP Y: 0	
Alleles at SNP X	1		11	10	p_1
		Actual	p_{11}	p_{10}	
		Expected	p_1q_1	p_1q_0	
	0		01	00	p_0
		Actual	p_{01}	p_{00}	
		Expected	p_0q_1	p_0q_0	
			q_1	q_0	1

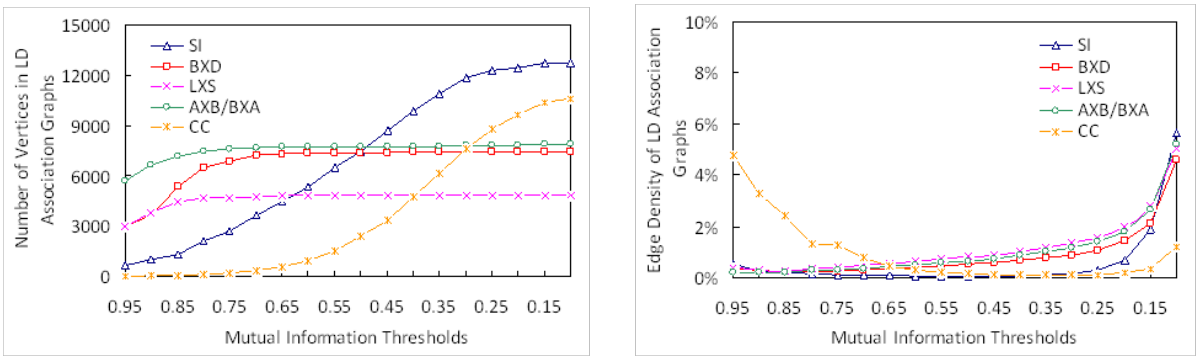
Construction of SNP-SNP Association Graph

Second, the mapping module applies a thresholding procedure and a graph transforming procedure on the LD coefficient matrix to construct unweighted graphs where vertices represent SNPs and edges represent SNP-SNP associations. A high-pass filter is first applied incrementally to transform the real-valued matrix to binary matrices. The filter consists of thresholds varying from 1 to 0 with a decreasing step of 0.05 so that LD coefficients are transformed at each threshold to 1s if greater than or equal to the threshold, or to 0s otherwise. The resulting binary matrices are then represented as unweighted graphs where a pair of SNPs has an edge between them if the value is 1, and no edge otherwise.

Figure 5.4 plots the number of vertices and the edge density of the SNP-SNP association graphs constructed for five mouse populations with thresholds ranging from 0.95 to 0.10 using mutual information. With the decreasing of the threshold, both the graph size and the edge density increase. The only exception is the edge density for the CC strains at high thresholds (> 0.7), where the graphs have less than 100 vertices. With the increase of the graph size and edge density, the problem complexity subsequently increases. The graphs constructed for the SI strains at low thresholds (< 0.3) have more than 12,000 vertices and less than 6% edge density.

Enumeration of Maximal Cliques

Third, the computational module applies clique-centric methods on the SNP-SNP association graph to enumerate maximal cliques using MCLEARS (see Chapter 3). Figure 5.5 (a) shows the numbers of maximal cliques increase exponentially with the decrease of the threshold. For SI, the number of maximal cliques at low thresholds shoots up when the graph size rises to more than 12,000 and the edge density exceeds 2%. Figure 5.5 (b) shows that the size of the largest clique increases gradually for all populations when allowing edges between SNPs with lower LD coefficients. The profile of maximal cliques indicates the extension of graphs enlarges the existing cliques and creates new ones as well. The enormous number of maximal cliques detected at low thresholds reflects the fact that these cliques are highly overlapped. Paraclique [19] is thus used to merge highly overlapped maximal cliques to form dense subgraphs, where a cutoff value is used to control the density of paracliques. The number of paracliques for these mouse populations is plotted in Figure 5.5 (c).



(a) (b)

Figure 5.4: Properties of SNP-SNP association graphs for five mouse populations with thresholds ranging from 0.95 to 0.10 using mutual information: (a) the number of vertices in the graph, (b) the edge density of the graph.

A maximal clique or paraclique represents a set of SNPs pairwisely associated with LD coefficients greater than a certain threshold, or called an *LD network*. The clique-based model allows the detection of LD networks with SNPs that are not closely located on a chromosome. An LD network may contain either SNPs located at multiple chromosomes, or contain a large number of SNPs closely located on a single chromosome.

Analysis of LD Networks

Last, the analytical module quantitatively measures the LD networks for the size distributions, the genome coverage in base pairs, and the chromosomal representation by members. LD networks in different populations are compared by their members to find those conserved in all populations or differed in various populations. LD networks can also be visualized as an LD map of genome using developed visualization tools.

To analyze the structure of LD networks, the *purity* of an LD network is measured as the number of the chromosomes on which SNPs in the network are located. Syntenic or non-syntenic LD networks can be identified depending on the purity of the LD networks. *Syntenic* LD networks consist of associated SNPs on the same chromosome and non-syntenic LD networks contain associated SNPs across different chromosomes. The hypothesis is that the non-random breeding history of standard inbred strains has resulted in the infiltration of non-syntenic linkage even at high thresholds, while most of the LD networks of randomly mated RI lines are short-range or long-range synthetic LD.

Figure 5.5 (d-f) shows the purity distribution of LD networks in different populations. The two-way RI sets have LD networks with relatively constant purity of 1 to 3 chromosomes, while SI strains have a gradual decrease of purity, which is readily apparent at LD threshold < 0.3. The CC strains have high purity, even at thresholds that reveal impure LD networks in SI strains. Results indicate that the genotype structure of SI strains that have had longer periods of out-crossing consists of smaller blocks of associated SNPs than RI strains. Large syntenic LD networks in RI sets, though relatively uncorrelated with other genome regions, limit the power and precision of this population for genetic analysis. The CC set, as it was designed, has both smaller syntenic LD networks than RI sets, and less non-syntenic LD networks than the SI strains.

5.1.3 Conclusion

In this section, the clique-based model with the MCLEARS algorithm was applied to the linkage disequilibrium analysis, which provided a convenient tool for quantitative measurement of population structure. Maximal cliques were enumerated by MCLEARS from graphs of various sizes and densities constructed at a variety of thresholds from large SNP sets. With the decrease of linkage threshold, the graph size and edge density increase; subsequently the problem complexity increased. Results showed that graphs constructed in this study can be large ($> 12,000$ vertices) but sparse ($< 6\%$ edge density). The number of maximal cliques found in the largest graph can go up to 1×10^{10}. Therefore, highly overlapped maximal cliques were merged into paracliques to represent LD networks. This method provided a tool not only to understand a population's genetic structure by analyzing the size, number, and location of SNPs in LD networks detected in this population, but also to identify conserved LD regions across populations by comparing SNPs in LD networks found in multiple populations.

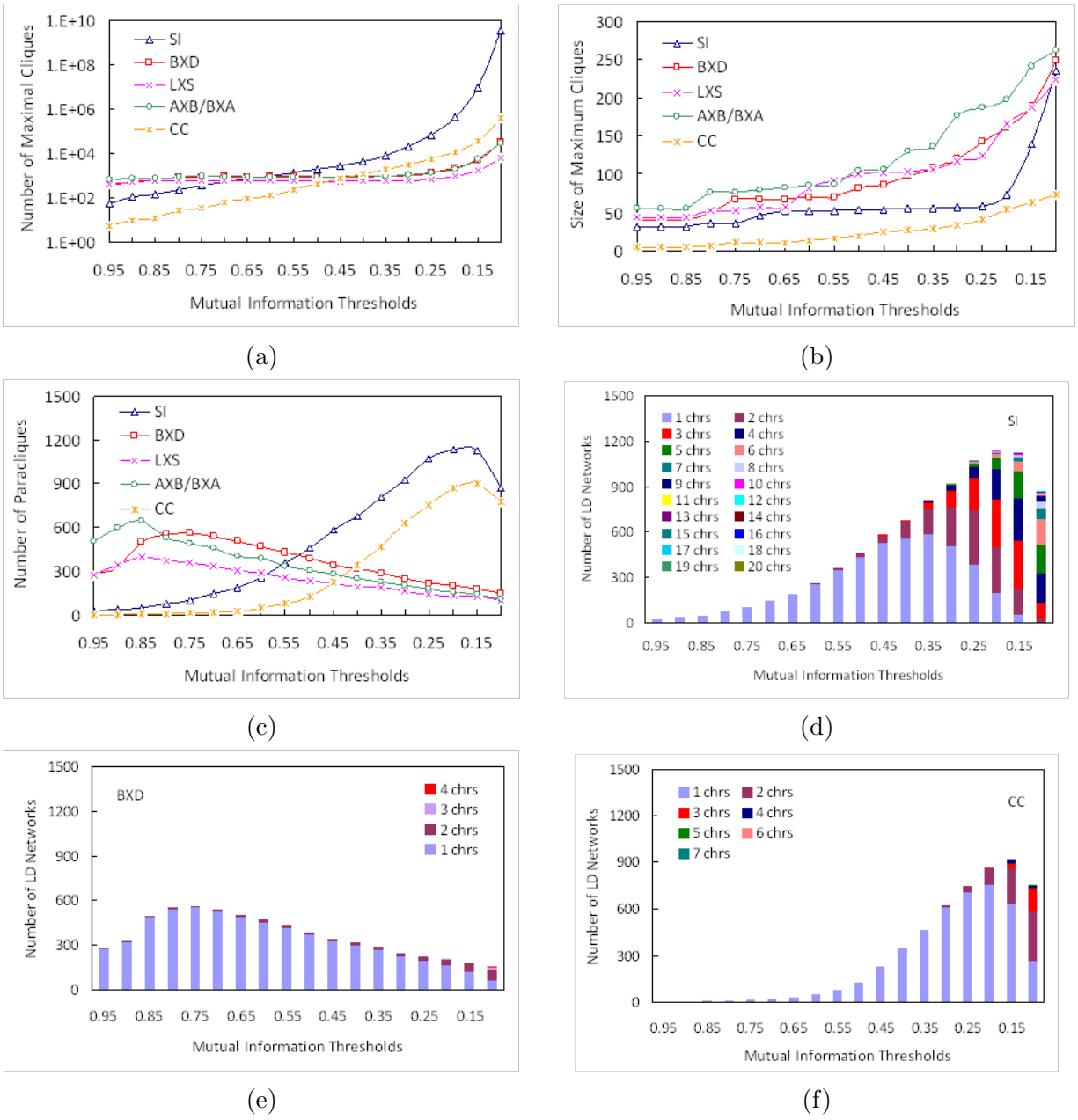


Figure 5.5: Properties of maximal cliques in SNP-SNP association graphs and sample analytical results of LD networks in five mouse populations with thresholds ranging from 0.95 to 0.10: (a) the number of maximal cliques, (b) the size of maximum cliques, and (c) the number of paracliques detected at each threshold; and (d-f) the purity of LD networks in SI, BXD, and CC strains using paracliques.

5.2 Application II: Ontological Discovery

The study of ontological discovery seeks to identify relations among genes and their roles in the biological processes. Using the graph model, the MBEA algorithm is applied to discover the ontology of biological processes based on the enumeration of maximal bicliques in gene-phenotype association graphs. This method integrates gene and phenotype data and represents the associations as a bipartite graph, which allows relationships from diverse experiments to be combined. Phenotypes associated with a common set of genes are computationally derived by the enumeration of bicliques using MBEA. Maximal bicliques are organized as a directed acyclic graph that forms an empirically derived representation of the relationships among biological processes and the genes that underlie these functions. An implementation of this method and MBEA has been embedded in the web-based Ontological Discovery Environment (ODE, `http://ontologicaldiscovery.org`) tool.

5.2.1 The Biological Problem

Background

All living organisms depend on DNA to pass on their traits to offspring. A *gene* is a segment of DNA sequence that must be inherited as a whole to define traits. It contains both coding sequences that determine what the gene does and non-coding sequences that determine when the gene is active (*expressed*). When a gene is expressed, the coding and non-coding sequences are copied and an *RNA* copy of the gene's information is produced during a process called *transcription*. This piece of RNA can then direct the synthesis of *proteins* via the genetic code in a process called *translation*. These molecules resulting from gene expression, whether RNA or protein, are known as gene products, and are responsible for the development and functioning of all living things. The transcription of DNA sequences to form RNA and the translation of RNA to form proteins is the Central Dogma of Biology.

The genetic encoding in an organism is its *genotype* and the resulting physical characteristics is its *phenotype*. Any observable characteristic of an organism is a phenotype. Examples include physical appearance such as height, weight, hair color, and eye color, and behavioral responses such as stress, addiction, learning, pain, and so on. A *trait* is a distinct phenotypic character of an organism that may be inherited, environmentally determined or somewhere in between. For example, human skin color is a phenotype; white, black, and yellow are traits; the particular set of genes an individual possesses that decides his or her skin color is his or her genotype. A trait controlled by a single gene is a *qualitative trait*. Most traits that are attributed to two or more genes and their interaction with the environment are *quantitative traits* or *complex traits*. Moreover, a phenotype of an individual is determined by not only its genotype but also by the environmental factors because it is the proteins that perform most life functions, and they are dynamically changed in response to environmental signals.

Motivation

A major challenge in bioinformatics is to identify relationships among genes and their varied roles in biological processes. A complex trait is attributed to multiple genes; a gene may be involved in multiple biological processes. When broadly designating any biological process as a phenotype, it is useful to categorize phenotypes based on the sets of underlying genes. Since set relationships are discrete structures that can be naturally described as finite simple graphs, graph algorithms can be utilized to interpret and analyze the enormous correlation matrices that arise in the study of genomic and transcriptomic data. To find associations between data of heterogeneous types, the bipartite

graph model with biclique algorithms become the appropriate choice. However, the biclique-based model for gene set analysis can be computationally critical because the classification and assessment of the phenome space is theoretically unbounded, especially for the genome-scale ontology discovery. The fast biclique algorithm MBEA was thus developed to address these computational demands for the creation of emergent phenotype ontology.

Data Sets

In this research, phenotype is loosely defined as any set of genes that comes from some biological source such as an experiment related to drug abuse. They can be generated through any methodology dedicated to gene-network creation. For example, gene sets may be defined from public microarray gene expression data, from genetic correlation to gene expression, from literature associations obtained via text mining tools, and from numerous literature reviews and empirical studies in which researchers have compiled gene lists involved in various behavioral constructs including pain, aggression, alcohol specific, and drug abuse.

The ODE database has 180 non-empty gene sets uploaded to date. These gene sets come from five species: *Drosophila Melanogaster* (fruit fly), *Danio Rerio* (zebrafish), *Homo Sapiens* (human), *Mus Musculus* (mouse), and *Rattus Norvegicus* (rat). When gene sets from various sources are combined, the *homology* is included to match genes with highly similar DNA sequences in different species. The combination of all 180 gene sets with the inclusion of homology results in a total number of 13,307 genes. Although the number of gene sets is relatively small in the current ODE database, the number can increase quickly when researchers start to use the tool and upload their gene sets. Although the phenome space (i.e. the total number of phenotypes) is theoretically unbounded, the genome space (i.e. the total number of genes) is constrained by the number of genes in the genome of human or other organisms; for example, the human genome is estimated to contain 20,000-25,000 genes.

5.2.2 The Biclique-Based Model

A biclique-based model was developed to extract functionally similar genes and phenotypes from gene sets derived from various sources and to organize them as a directed acyclic graph (DAG) to represent the ontology of phenotypes. Figure 5.6 is an overview of the biclique-based graph model for this application. The model consists of three major components: (a) a statistical module for the computation of gene-phenotype association matrix and the construction of bipartite graphs by thresholding and graph mapping, (b) a computational module for the enumeration of maximal bicliques from the bipartite association graphs using the MBEA algorithm (see Chapter 4), and (c) an analytical module for the organization of gene sets as phenotype ontology by integrating maximal bicliques into a DAG.

Construction of Gene-Phenotype Association Graph

First, the statistical module combines gene sets from various sources, computes a matrix of scores for the associations of genes to phenotypes, transforms the score matrix into a binary matrix by applying a threshold on the scores, and then maps the binary matrix to a bipartite graph as inputs to the computational module. A matrix of scores for the associations of genes to phenotypes can be computed by several statistical metrics, such as correlation coefficients, p-values, q-values, or by binary values in instances of curated literature associations or other categorical analyses. Low-pass (for p-values, q-values) or high-pass (for correlations) filtering thresholds are then used to transform the matrices into binary matrices, which are represented as bipartite graphs with

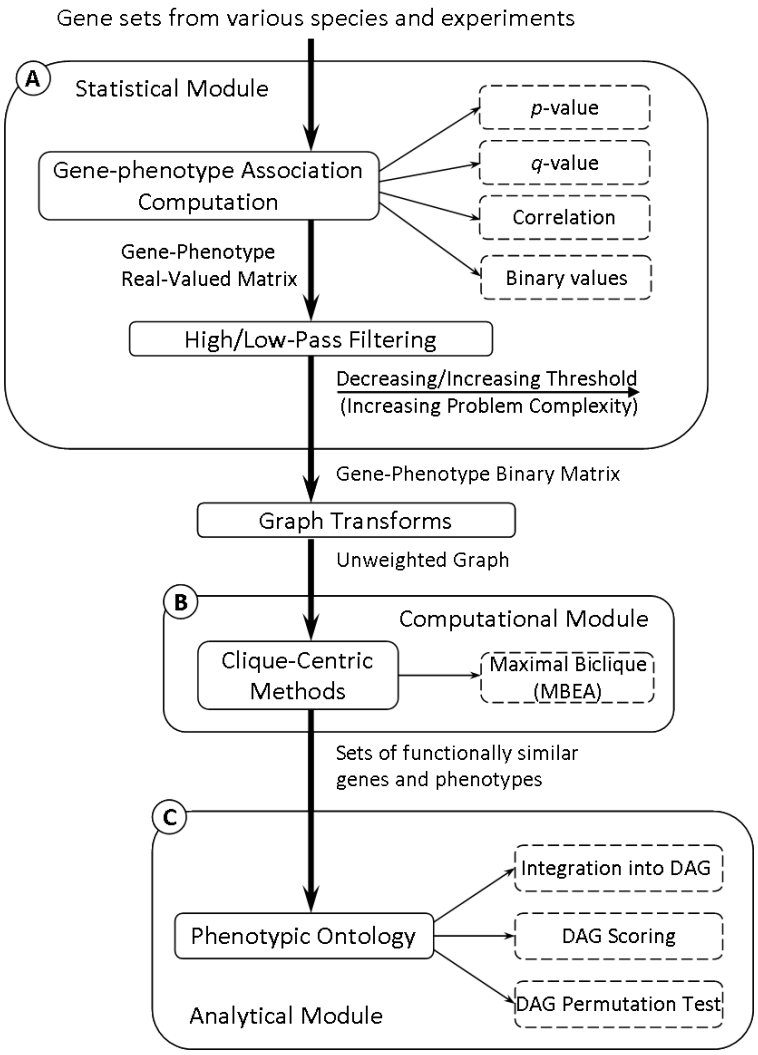


Figure 5.6: An overview of the biclique-based model for phenotypic ontology on gene sets of heterogeneous data types. The model include (A) a statistical module for computing the gene-phenotype association matrix, thresholding, and constructing unweighted graphs, (B) a computational module for detecting sets of functionally similar genes and phenotypes using MBEA, and (C) an analytical module for integrating maximal bicliques into a DAG that represents the ontology of phenotypes.

two disjoint sets of vertices representing genes and phenotypes separately and edges representing associations of genes to phenotypes. This representation allows us to construct heterogeneous data sets as a gene-phenotype association graph and to apply MBEA to discover maximal bicliques.

Table 5.3 lists the size and density of gene-phenotype association graphs constructed for gene sets selected from the ODE database. These graphs are constructed for gene sets selected using different keywords, such as "drug," "alcohol," and "cerebellum," from all species. These sets of gene sets were also combined to construct larger phenotype sets. For each phenotype set, gene sets were combined including homology, and p-values were computed as significant scores of gene-phenotype associations. The score matrix was transformed to a binary matrix using threshold 0. This means that any association will be considered when constructing the bipartite graph. Such a threshold was chosen in order to make graphs as large and dense as possible because of the small number of available gene sets. The largest graph is the one constructed for all gene sets; it has 180 phenotypes, 13,307 genes, and 1.45% edge density.

Enumeration of Maximal Bicliques

Second, the computational module enumerates all maximal bicliques in the gene-phenotype association graph using MBEA. Each maximal biclique corresponds to a set of phenotypes and the common genes to which they are all associated where the set of genes is not contained in other phenotypes. A maximal biclique thus represents a set of functionally similar biological processes and the set of common genes underlying these phenotypes. A sequential implementation of MBEA is currently fast enough for ODE due to the relatively small number of uploaded gene sets. Table 5.3 shows the number of maximal bicliques found in these gene-phenotype association graphs and run times using an implementation of MBEA performed on a Dell OptiPlex with Intel Core 2 Quad 2.5GHz processors and 4GB DDR2. For all graphs, MBEA found all maximal bicliques within a second. The short run time is attributed to both the fast MBEA implementation and the relatively small and sparse graphs. The MBEA implementation ensures ODE to provide online services to researchers. With the increasing number of gene sets uploaded to ODE, parallel implementations of MBEA may be needed to provide the same speed.

Integration of Maximal Bicliques into Phenotype Ontology

Last, the analytical module constructs the ontology of phenotypes by integrating the maximal bicliques into a DAG of phenotype relationships. A phenotype DAG is an is_a hierarchy for the classification of phenotypes. Maximal bicliques enumerated from the gene-phenotype association graph are integrated into a directed acyclic graph of phenotypes based on their relationships. DAGs are similar to hierarchies but differ in that a child node can be related to multiple (≥ 1) parent nodes. The formulation of the phenotype DAG is based on Observation 9, an inherent ordering of the bicliques.

Observation 9 Let $P(b)$ be the phenotype set of a biclique b and $G(b)$ be the intersection of genes. Given two maximal bicliques b_1 and b_2, $P(b_1) \supset P(b_2)$ if and only if $G(b_1) \subset G(b_2)$, and $P(b_1) \subset P(b_2)$ if and only if $G(b_1) \supset G(b_2)$.

A phenotype DAG is thus a collection of maximal bicliques and a partial ordering of the bicliques. The maximal bicliques become nodes, and the hierarchy is defined by the subset relationships between their phenotype sets. A node b_1 is defined as an *ancestor* of node b_2 if $P(b_1) \supset P(b_2)$ and is a *descendant* of b_2 otherwise. Node b_1 is called a *parent* of b_2 if there is no node b_3 existing such that b_3 is both a descendant of b_1 and an ancestor of b_2. Correspondingly, b_2 is a *child* of b_1. In

Table 5.3: The graph size, the number of maximal bicliques, and run times using MBEA for bipartite graphs constructed for gene sets selected from the ODE database.

Sets	#Phenotype	#Gene	#Edge	Density	#Biclique	Run times (Seconds)
Drug	14	1784	2795	11.2%	36	0.007
Alcohol	26	2922	3856	5.08%	79	0.014
Cerebellum	51	4541	8788	3.79%	166	0.031
A&C	73	6031	11535	2.62%	436	0.053
D&A&C	80	6802	13469	2.48%	634	0.062
All	180	13307	34841	1.45%	6604	0.309

a DAG, a node can have multiple parent nodes, not limited to two even though the term "parent" is used. Edges exist only from a node to its child nodes. The formal definition of a phenotype DAG is as follows:

Definition 3 Let G_B be a bipartite graph, and $B(G_B)$ be the set of all maximal bicliques in G_B. A phenotype DAG $G_P = (V, E)$ of G_B is a directed acyclic graph, where $V = \{v \mid v \in B(G_B)\}$, and $E = \{(v_1, v_2) \mid v_1, v_2 \in V,\ P(v_1) \supset P(v_2),\ \text{and}\ \nexists\ v_3 \in V,\ P(v_1) \supset P(v_3) \supset P(v_2)\}$.

*Root node*s in a phenotype DAG are defined as those with no ancestor, i.e. $R = \{v \mid v \in V(G_P),\ \nexists\ u \in V(G_P),\ P(u) \supset P(v)\}$, and *leaf node*s are those with no descendant, i.e. $T = \{v \mid v \in V(G_P),\ \nexists\ u \in V(G_P),\ P(u) \subset P(v)\}$. Phenotype sets that reside in the root nodes are phenotype supersets. Phenotype subsets in the descendant nodes are connected to additional genes. The largest possible size of a phenotype superset is the total number of phenotypes to be categorized. A set of pairwise disjoint phenotypes results in a DAG of smallest size consisting only a set of leaf nodes with a single phenotype.

It should be noted that the total possible nodes of a phenotype DAG is the total number of 2^n-1 subsets for a set of n phenotypes. The number of maximal bicliques (i.e. the number of observed DAG nodes) can also be enormous for large and dense bipartite graphs. Therefore, node splitting rules based on similarity and stopping rules based on node size can be applied to limit the growth and density of the phenotype DAG. Moreover, several metrics can be used to describe the information of a phenotype DAG, such as the number of root nodes, the length of the longest path among all paths from a root node to a terminal node, and the percent of observed DAG nodes in all possible nodes. A re-sampling procedure can be performed to test the significant level of the DAG given the same number of phenotypes, genes, and associations.

An Example

Figure 5.7 illustrates the creation of the phenotype DAG for four gene sets (phenotypes) that have binary associations to a total number of eight genes (Figure 5.7 (a)). These four phenotypes with their genes are first constructed as a gene-phenotype association graph (Figure 5.7 (b)). A total number of eight maximal bicliques are then enumerated in the association bipartite graph using MBEA, three of which are highlighted in Figure 5.7 (c). They are then integrated to form a phenotype DAG as shown in Figure 5.7 (d). Maximal bicliques $(\{1, 2, 3\}, \{a, b\})$ (dashed lines) and $(\{2, 3, 4\}, \{d\})$ (dotted lines) are two root nodes in the DAG. Maximal biclique $(\{2, 3\}, \{a, b, d, f\})$ (dotted dash line) is a child node of both root nodes, since $\{2, 3\}$ is a subset of phenotype sets in both roots. Phenotype subset $\{2, 3\}$ is associated with the same genes a, b and d as its parents, but is also connected to one additional gene f.

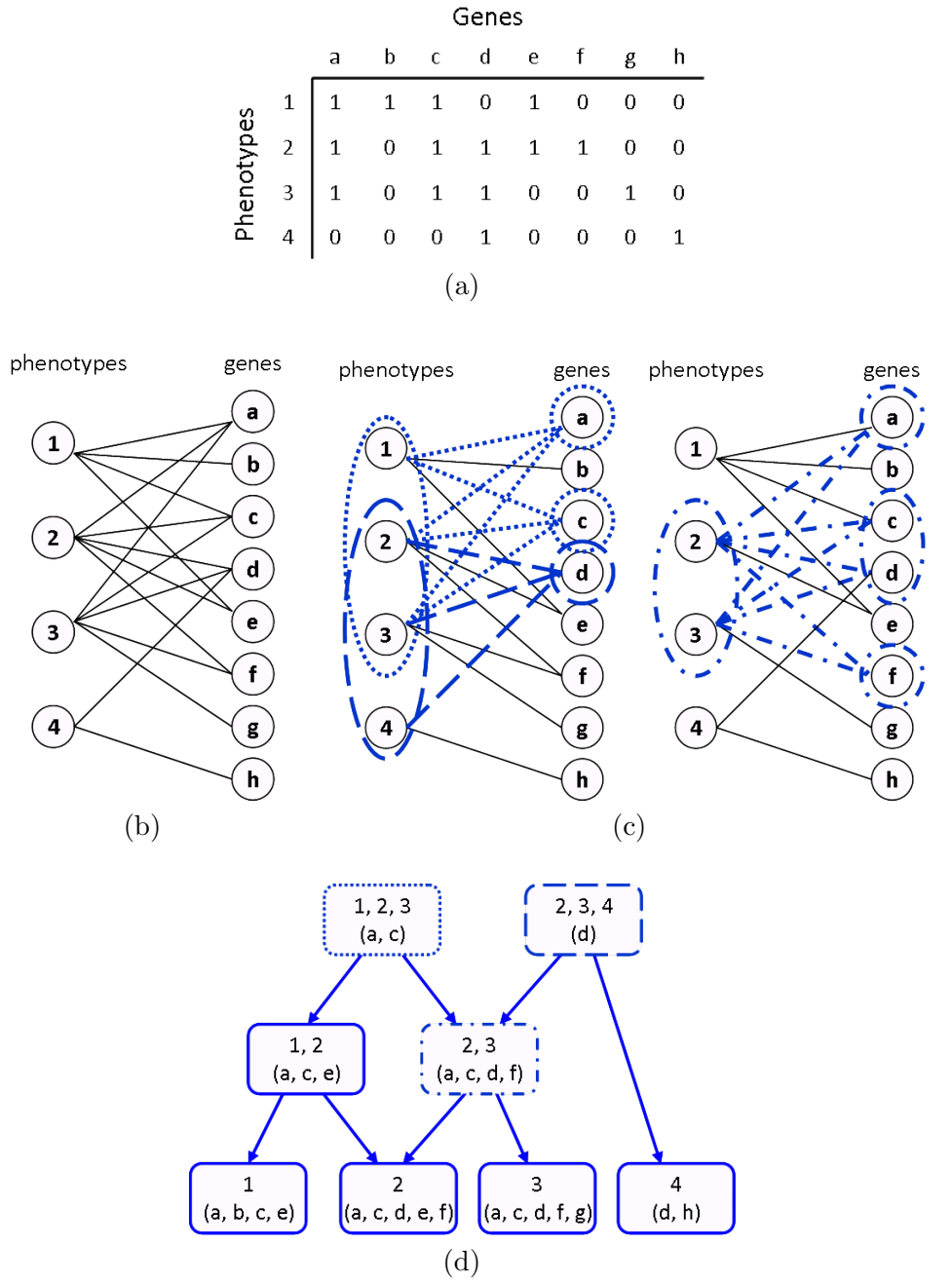


Figure 5.7: Creation of DAG for phenotypic ontology: (a) four gene sets (phenotypes) with binary associations to a total number of 8 genes, (b) gene-phenotype bipartite graph constructed, (c) three maximal bicliques in the association graph, and (d) integration of all maximal bicliques to a DAG of phenotype relationships.

5.2.3 Conclusion

This section showed the utility of the biclique-based model and the MBEA algorithm in the study of phenotypic ontology. The fast implementation of MBEA made it possible for the web-based ODE tool to provide online classification of large number of gene sets. MBEA can enumerate maximal bicliques within a second for gene sets in ODE database to date. MBEA also ensured the fast speed to solve large bipartite graphs when increasing number of gene sets uploaded to ODE by biologists. Furthermore, in order to compute the significant level of DAG scores (e.g. height, width) for a given set of gene sets among sets with the same number of phenotypes, genes and positive associations, a re-sampling procedure can be applied to the set of gene sets to simulate different gene-phenotype associations. Such a procedure may require thousands or tens of thousands times re-sampling. Without a fast biclique algorithm such as MBEA, it is impossible to finish the significant test within an hour for moderate number (e.g. hundreds) of gene sets.

Chapter 6

Summary and Discussion

This dissertation focuses on the design and implementation of exact, efficient, and scalable algorithms for clique enumeration problems that are computation and memory intensive and the applications of these algorithms in the elucidation of large-scale biological networks. The *Clique Enumerator* is developed to provide solutions for the maximal clique problem. It uses fixed parameter tractable implementations for the maximum clique problem to decide a reasonable upper bound (ub) of clique sizes and uses the k-clique enumeration algorithm to find both maximal and non-maximal cliques of a user-supplied lower bound (lb). After that, the MCLEARS algorithm is used to enumerate maximal cliques within the range [lb, ub], taking non-maximal lb-cliques as inputs. MCLEARS is able to enumerate maximal cliques in non-decreasing order, which allows the user to track the progress of algorithm execution. MCLEARS uses a compact bitmap data representation to reduce memory usage and to enable very fast bitwise operations for checking clique maximality. The MCLEARS algorithm is paralleled on ultra-large shared memory machines to lower the memory requirement on a single processor and to achieve linear speedups with dynamic load balancing techniques.

The memory management techniques used in *Clique Enumerator* can be applied to other exact algorithms for difficult combinatorial problems. These algorithms frequently use recursive backtracking and model their search space by a tree in which a depth-first traversal is used to identify optimum solutions. Due to large input instances, the effectiveness of these algorithms relies heavily on judicious memory management. Dynamic programming problems can also be considered. In this context, we generally trade space for time. In the case of $\mathcal{NP}$-complete problems in particular, the time saved is offset by gigantic storage requirements. Paging, thrashing, and other potential side-effects of out-of-core implementations are apt to negate the time savings expected with dynamic programming unless memory intensive management techniques are applied.

The MBEA algorithm is the first general-purpose algorithm to enumerate all maximal bicliques in a bipartite graph. Its efficiency is achieved by exploiting the inherent structure of bipartite graphs, and by applying branch-and-bound techniques to reduce the search space. Experimental results on both real biological data and synthetic data show that the algorithm is significantly faster than its best competitors. This is not at all surprising given that no general-purpose algorithm has previously been published and analyzed. The algorithm is further improved by a method to remove dominated vertices from the candidate set at a earlier state and a method to select vertices from the candidate set based on their degrees. Both methods prune out more non-maximal candidates. Experimental results show that the improved algorithm is faster than the basic version in biological bipartite graphs that have diverse vertex degrees.

A sequential version of the MBEA algorithm has been employed in the web-based "Ontological Discovery Environment" tool to generate a Phenome Independence and Similarity Hierarchy graph for ontology discovery. The PhISH graph is constructed using maximal bicliques derived from a gene-phenotype association graph. The sequential implementation is fast enough to support current gene sets analysis due to the relatively small number of uploaded gene sets. With the increasing number of gene sets from various sources, a parallel implementation might be of interest either on shared memory machines using multithreading or on distributed memory machines using MPI (Message Passing Interface). The framework of the parallel MCLEARS algorithm can be reused for the implementation on shared-memory machines. Parallel algorithms on distributed-memory architectures should carefully consider load balancing among multiple processors due to the unbalanced nature of search trees.

Clique enumeration algorithms have been playing an important role in computational biology. We have shown their utilities in two novel applications in genetics: linkage disequilibrium network analysis and phenotypic ontology discovery. Clique algorithms can be applied to more biological applications that depend on the identification and analysis of networks of related biological objects by mapping objects as vertices and their relations as edges or edge-weights. Examples of biological objects vary from very small units, such as SNPs, to larger ones, such as proteins, including DNA sequences, genes, phenotypes, metabolites, protein secondary structure elements, species, and even individuals. Examples of biological relations include spatial and angular distances, correlations, interactions, regulations, and associations between objects of same type or of heterogeneous types. Examples of biological networks include protein complexes that are sets of pairwise interacted proteins, gene clusters that are networks of co-regulated genes, and phylogenetic data sets that are sets of genes sampled from multiple species. With exact, scalable, and efficient graph algorithms, such networks can be extracted from large-scale data sets.

Bibliography

Bibliography

[1] F. N. Abu-Khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova. On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput computational biology. In *Proceedings, International Conference on Research Trends in Science and Technology*, 2005. 2, 11, 12, 16, 17, 20, 21, 22, 24, 34

[2] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings, Workshop on Algorithm Engineering and Experiments,* New Orleans, Louisiana, 2004. 22

[3] F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag. Scalable parallel algorithms for difficult combinatorial problems: A case study in optimization. In *Proceedings, International Conference on Parallel and Distributed Computing and Systems*, 2003. 12, 22

[4] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 45(3):269–284, July 2006. 12

[5] E. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2:1–6, 1973. 10, 11

[6] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Appl. Math.*, 145(1):11–21, 2004. 7, 14, 16, 43, 46

[7] P. J. Artymiuk, H. M. Grindley, A. R. Poirrette, D. W. Rice, E. C. Ujah, and P. Willett. Identification of beta-sheet motifs, of psi-loops and of patterns of amino acid residues in three-dimensional protein structures using a subgraph-isomorphism algorithm. *Journal of Chemical Information and Computer Sciences*, 34(1):54–62, 1994. 2

[8] J. Auguston and J. Minker. An analysis of some graph theoretical cluster techniques. *Journal of the ACM*, 17:571–588, 1970. 9, 10

[9] V. Bafna, D. Gusfield, S. Hannenhalli, and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *J. Computational Biology*, 11(5):858–866, 2004. 2

[10] N. E. Baldwin, E. J. Chesler, S. Kirov, M. A. Langston, J. R. Snoddy, R. W. Williams, and B. Zhang. Computational, integrative and comparative methods for the elucidation of genetic co-expression networks. *Journal of Biomedicine and Biotechnology*, 2:172–180, 2005. 2, 16, 17, 34

[11] N. E. Baldwin, R. L. Collins, M. A. Langston, M. R. Leuze, C. T. Symons, and B. H. Voy. High performance computational tools for motif discovery. In *Proceedings, IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, 2004. 2, 17, 20, 21

[12] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. *Handbook of Combinatorial Optimization*, 4:1–74, 1999. 9, 16

[13] R. E. Bonner. On some clustering techniques. *IBM Journal of Research and Development*, 8(1):22–32, 1964. 9

[14] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973. 10, 11, 21, 22, 38

[15] E. Cáceres, S. W. Song, and J. L. Szwarcfiter. A coarse-grained parallel algorithm for maximal cliques in circle graphs. In *ICCS '01: Proceedings of the International Conference on Computational Science-Part II*, pages 638–647, London, UK, 2001. Springer-Verlag. 12

[16] L. S. Chandran and F. Grandoni. Refined memorisation for vertex cover. In *Proceedings, International Workshop on Parameterized and Exact Computation*, 2004. 21

[17] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 93–103. AAAI Press, 2000. 13, 18

[18] E. J. Chesler, R. Kirova, E. J. Baker, Z. Li, A. D. Perkins, and M. A. Langston. Mapping the phenome space using combinatorial analysis of the empirical associations of genes and phenotypes. In *Proceedings, International Mammalian Genome Conference*, 2006. 18

[19] E. J. Chesler and M. A. Langston. Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data. Report 575, University of Tennessee, 2006. 3, 13, 16, 17, 49, 62

[20] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. W. J. Gu and, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, J. B. Hogenesch, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37:233–242, 2005. 16, 17, 21, 34

[21] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14:210–223, 1985. 10, 11, 12

[22] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, New York, 1999. 21

[23] N. Du, B. Wu, L. Xu, B. Wang, and X. Pei. A parallel algorithm for enumerating all maximal cliques in complex network. In *ICDMW'06: Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 320–324, Washington, DC, USA, 2006. IEEE Computer Society. 12

[24] D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, 1994. 5, 14, 15, 16, 43

[25] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms & Applications*, 7(2):131–140, 2003. 10, 11

[26] D. Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. In *Proceeding, 16th ACM-SIAM Symp. Discrete Algorithms*, pages 451–459, Vancouver, 2005. 10

[27] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1987. 21

[28] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35:727–739, 1988. 21

[29] M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. *Journal of Computer and Systems Sciences*, 49:769–779, 1994. 21

[30] D. Fernndez-Baca. The perfect phylogeny problem. *Steiner Trees in Industry*, 2002. 21

[31] E. Gardiner, P. Willett, and P. Artymiuk. Clique-theoretic techniques for macromolecular docking. *Computing*, 21:295–322, 1979. 2, 16, 17

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, New York, 1979. 4, 5, 9

[33] L. Gerhards and W. Lindenberg. Clique detection for nondirected graphs: Two new algorithms. *Computing*, 21:295–322, 1979. 10, 11

[34] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *Proceedings, FIMI'03: Workshop on Frequent Itemset Mining Implementations*, 2003. 15, 16, 46

[35] H. J. Greenberg, W. E. Hart, and G. Lancia. Opportunities for combinatorial optimization in computational biology. *INFORMS J. on Computing*, 16(3):211–231, 2004. 2

[36] H. Grindley, P. Artymiuk, D. Rice, and P. Willet. Identification of tertiary sturcture resemblance in proteins using a maximal common subgraph isomorphism algorithm. *Journal of Molecular Biology*, 29:707–721, 1993. 16

[37] E. Harley, A. Bonner, and N. Goodman. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics*, 17(6):487–494, 2001. 2, 10, 11, 16, 17, 21

[38] E. R. Harley. *Graph algorithms for assembling integrated genome map.* PhD thesis, University of Toronto, 2003. 9, 10, 11, 16

[39] A. Harrison, F. Pearl, I. Sillitoe, T. Slidel, R. Mott, J. Thornton, and C. Orengo. Recognizing the fold of a protein structure. *Bioinformatics*, 19(14):1748–1759, 2003. 16

[40] D. S. Johnson and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. 9, 10, 44

[41] H. Johnston. Cliques of a graph - variations on the bron-kerbosch algorithm. *International Journal of Computer & Information Sciences*, 5:209–238, 1976. 9, 10, 11

[42] H. Kato and Y. Takahashi. SS3D-P2: a three-dimensional substructure search program for protein motifs based on secondary structure elements. *Bioinformatics*, 13(6):593–600, 1997. 2, 16

[43] R. Kirova, M. A. Langston, X. Peng, A. D. Perkins, and E. J. Chesler. A systems genetic analysis of chronic fatigue syndrome: Combinatorial data integration from SNPs to differential diagnosis of disease. In *Proceedings, International Conference for the Critical Assessment of Microarray Data Analysis (CAMDA06)*, Durham, North Carolina, June 2006. 3, 13, 18

[44] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250:1–30, 2001. 10, 11

[45] I. Koch, T. Lengauer, and E. Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. *Journal of Computational Biology*, 3:289–306, 1996. 16

[46] F. Kose, W. Weckwerth, T. Linke, and O. Fiehn. Visualizing plant metabolomic correlation networks using clique – metabolite matrices. *Bioinformatics*, 17:1198–1208, 2001. 2, 7, 10, 11, 12, 16, 17, 20, 24, 26, 34

[47] E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Generating all maximal independent sets, NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9:558–565, 1980. 10

[48] R. Lewontin. The interaction of selection and linkage II: Optimum models. *Genetics*, 50:757–782, 1964. 60

[49] J. Li, H. Li, D. Soh, and L. Wong. A correspondence between maximal complete bipartite subgraphs and closed patterns. In *PKDD*, pages 146–156. Springer-Verlag, Berlin Heidelberg, 2005. 15

[50] G. Liu, K. Sim, and J. Li. Efficient mining of large maximal bicliques. In *The 8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2006)*, pages 437–448, 2006. 14, 16

[51] J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 187, Washington, DC, USA, 2003. IEEE Computer Society. 18

[52] E. Loukakis. A new backtracking algorithm for generating the family of maximal independent sets of a graph. *Computers & Mathematics with Applications*, 9:583–589, 1983. 10

[53] E. Loukakis and C. Tsouros. A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically. *Computing*, 27:249–266, 1981. 10, 11, 12

[54] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Proceeding, 9th Scandinavian Workshop Algorithm Theory (SWAT)*, pages 260–272, 2004. 10, 11, 13, 14

[55] W. Meeusen and L. Cuyvers. Clique detection in directed graphs: A new algorithm. *Journal of Computational and Applied Mathematics*, 1(3):185–193, 1975. 10, 11

[56] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965. 1, 9, 20

[57] R. A. Mushlin, A. Kershenbaum, S. T. Gallagher, and T. R. Rebbeck. A graph-theoretical approach for pattern discovery in epidemiological research. *IBM Systems Journal*, 46(1):135–149, 2007. 3, 13, 14, 16, 18

[58] R. Osteen. Clique detection algorithms based on line addition and line removal. *SIAM Journal on Applied Mathematics*, 26(1):126–135, 1974. 10

[59] R. Osteen and J. Tou. A clique-detection algorithm based on neighborhoods in graphs. *International Journal of Computer & Information Sciences*, 2:257–268, 1973. 9, 10, 11

[60] P. Pardalos, J. Rappe, and M. Resende. An exact parallel algorithm for the maximum clique problem. In R. D. Leone, A. Murl'i, P. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Optimization*, pages 279–300. Kluwer Academic Publishers, 1998. 12

[61] P. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994. 9

[62] R. Peeters. The maximum edge biclique problem is np-complete. *Discrete Appl. Math.*, 131(3):651–654, 2003. 5

[63] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521–533, 2002. 2

[64] J. M. Robson. Finding a maximum independent set in time $o(2^{n/4})$. Report 1251-01, LaBRI, Universite Bordeaux, 2001. 21

[65] 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. 3, 13, 14, 16, 18

[66] S. Schmitt, D. Kuhn, and G. Klebe. A new method to detect related function among proteins independent of sequence and fold homology. *Journal of Molecular Biology*, 323:387–406, 2002. 16

[67] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18:S136–S144, 2002. 2, 13, 16, 18

[68] B. Teusink, F. Baganz, H. V. Westerhoff, and S. G. Oliver. Metabolic control analysis as a tool in the elucidation of the function of novel genes. *Methods in Microbiology*, 26:297–336, 1998. 17

[69] K. K. Tomczak, V. D. Marinescu, M. F. Ramoni, D. Sanoudou, F. Montanaro, M. Han, I. S. K. L. M. Kunkel and, and A. H. Beggs. Expression profiling and identification of novel genes involved in myogenic differentiation. *FASEB Journal*, 18:403–405, 2004. 34

[70] E. Tomita, A. Tanaka, and H. Takahashi. An optimal algorithm for finding all the cliques. Report 1989-AL-12, IPSJ, 1989. 11

[71] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques. In *Proceedings, Computing and Combinatorics Conference*, 2004. 10

[72] S. Tsukiyama, H. A. M. Ide, and I. Shirakawa. A new algorithm for generating all the maximum independent sets. *SIAM Journal on Computing*, 6:505–517, 1977. 10, 11, 12

[73] T. Uno, M. Kiyomi, and H. Arimura. LCM ver.2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings, FIMI'04: Workshop on Frequent Itemset Mining Implementations*, Brighton UK, November 2004. 15

[74] C. von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002. 18

[75] B. H. Voy, J. A. Scharff, A. D. Perkins, A. M. Saxton, B. Borate, E. J. Chesler, L. K. Branstetter, and M. A. Langston. Extracting gene networks for low-dose radiation using graph theoretical algorithms. *PLoS Computational Biology*, 2(7):e89, 2006. 16, 17

[76] L. Wan, B. Wu, N. Du, Q. Ye, and P. Chen. A new algorithm for enumerating all maximal cliques in complex network. In *LNAI 4093, Advanced Data Mining and Applications*, pages 606–617. Springer Berlin Heidelberg, 2006. 10, 11, 12

[77] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 394–405. ACM Press, 2002. 13, 18

[78] J. Wang, J. Pei, and J. Han. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings, the 9th ACM SIGKDD Conference*, pages 236–245, 2003. 15

[79] N. Weskamp, D. Kuhn, E. Hullermeier, and G. Klebe. Efficient similarity search in protein structure database by k-clique hashing. *Bioinformatics*, 20(10):1522–1526, 2004. 16

[80] H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. 2, 16, 18

[81] M. J. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings, SIAM International Conference on Data Mining*, pages 398–416, 2002. 15

[82] M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *Proceedings, 3rd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1998. 14, 15

[83] Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proceedings, Supercomputing*, 2005. 7, 20

[84] Y. Zhang, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm with application to the integration of diverse biological data types. In *Proceedings, HICSS*, Big Island, Hawaii, January 2008. 7

[85] Y. Zhang, R. Kirova, G. A. Churchill, M. A. Langston, and E. J. Chesler. Comparison of genetic structure of mouse populations by combinatorial analysis of high-order long-range linkage disequilibrium networks using high density genotypes. In *Posters, UT-ORNL-KBRIN Bioinformatics Summit*, Buchanan, TN, April 2007. 7

[86] J. Zhu and G. Grahne. Reducing the main memory consumptions of FPmax* and FPclose. In *Proceedings, FIMI'04: Workshop on Frequent Itemset Mining Implementations*, Brighton UK, November 2004. 7, 15, 16, 46

Vita

Yun Zhang was born in Hangzhou, China, on October 3, 1976. After graduating from the Affiliated Middle School of Nanjing Normal University, Nanjing, China in 1994, she attended Nanjing University of Post and Telecommunications in Nanjing, China, where she received a Bachelor of Engineering degree in 1998 and a Master of Engineering degree in 2001 from the Department of Computer Science and Technology. She then attended the University of Texas at Dallas for graduate school and received a Master of Science degree in Computer Science in 2003. In the Spring of 2004, Yun started her doctoral study at the University of Tennessee in Computer Science. In Fall 2004, she joined the Graph Theory group as a graduate research assistant where she completed her Doctor of Philosophy degree in 2008. During her Ph.D. study, she worked as a research assistant in Computer Science and Mathematics Division of Oak Ridge National Laboratory from 2004 to 2006, and then in Bioscience Division since 2006.