NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Fast Algorithms For Finding The Maximum Edge Cardinality Biclique In Convex Bipartite Graphs

by

Shuye Pu

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Masters of Science
in Information and Systems Science

School of Computer Science
Carleton University
Ottawa, Ontario, Canada

June 10, 2004

© Copyright By Shuye Pu, 2004

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97480-4

Our file Notre référence

ISBN: 0-612-97480-4

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

Abstract

The problem of finding the maximum edge cardinality biclique (MECB) in convex bipartite graphs has been studied in the context of biclustering analysis of gene expression data matrices. A bipartite graph $G = (A, B, E)$, where A and B represent the bipartitions of the vertex set and E denotes the edge set, is convex on B if there is an ordering of B such that the neighborhood of each vertex in A forms a consecutive interval on B . A biclique is a complete bipartite subgraph. We found that the number of maximal bicliques in a B -convex bipartite graph is $O(|A|^2)$. We designed and analyzed two algorithms, BICLIQUE_DOMA and BICLIQUE_DOM, to find MECB in a convex bipartite graph. Algorithm BICLIQUE_DOMA takes $O(d_{\max}(B)|A|)$ time and uses $O(|A|)$ space, where $d_{\max}(B)$ is the maximum degree of B ; with minor modifications, this algorithm can also be used to list all maximal bicliques in convex bipartite graphs. Algorithm BICLIQUE_DOM runs in time $O(|B|^2 + |A|)$ and requires $O(|B| + |A|)$ space. We also designed and analyzed algorithms BICLIQUE_MAAPR and BICLIQUE_PERM for finding MECB in doubly convex bipartite graphs and bipartite permutation graphs, respectively. While BICLIQUE_MAAPR runs almost in time linear in $|A|$, BICLIQUE_PERM's run time is linear in $|A|$, which is optimal. Both algorithms use $O(|A|)$ storage space. These algorithms were implemented and tested subsequently. Our experimental results agree with those of the theoretical analysis. These algorithms represent the best solutions up-to-date to the problem of finding MECB in convex bipartite graphs.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisors, Dr. Doron Nussbaum and Dr. Jörg-Rüdiger Sack, for their unparalleled academic guidance and financial support, which allowed me to pursue a Masters degree in full time while raising two lovely kids. I can never thank them enough for their precious time spent on discussions that helped me to crystallize some key ideas in algorithm design and analysis. By the way, you would agree with me that those discussion sessions were truly enjoyable if you were there. I am so fortunate to have both of them as my mentors at the same time. The great help from administrative staff of the School of Computer Science is highly appreciated. I would like to thank Linda Pfeiffer, Jane MacArthur and Sandy Dare for their cordial assistance.

I also would like to thank fellow students in the Paradigm Research Laboratory for their help and support. Special mentioning goes to Ji Chen, Rob Kane, Hua Guo, Ting Wen and Mary Wang. I will never forget the following people who teamed up with me as Teaching Assistants: David Jackson, Marcin Pilat, Yihui Tang and Deepak Saraswat.

The financial supports from the National Science and Engineering Research Council of Canada in the form of an NSERC Scholarship, and from Carleton University in the form of Bursary, Research Assistantship and Teaching Assistantship, are gratefully acknowledged.

Last but not least, special thanks go to my lovely wife, Yumei Tang, for her moral support and selfless contribution to our family.

Table of Contents

ABSTRACT	III
ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS	V
LIST OF FIGURES	IX
LIST OF TABLES	X
CHAPTER 1 INTRODUCTION.....	1
1.1 Microarray data matrix and biclustering analysis	1
1.2 The DNA Microarray data modelling using bipartite graph.....	3
1.3 Motivation of the study.....	4
1.4 Contributions of the study.....	5
1.5 Thesis organization	6
CHAPTER 2 FOUNDATION OF THIS STUDY.....	7
2.1 Preliminaries	7
2.2 Existing results	11

2.2.1 Layered range tree related algorithms.....	11
2.2.1.1 Construction of a layered range tree	12
2.2.1.2 Dominance counting query on a layered range tree.....	15
2.2.2 Totally monotone matrix and the SMAWK algorithm	17
2.2.3 Totally monotone partial matrices and related searching algorithms	21
2.3 Related work.....	27
2.3.1 The consensus algorithms	28
2.3.1.1 The basic consensus algorithm	29
2.3.1.2 The modular consensus algorithm	29
2.3.1.3 The modular input consensus algorithm	30
2.3.2 Enumeration of all maximal bicliques in chordal bipartite graphs	31
2.3.2.1 Perfect edges without vertex elimination ordering (<i>PEWVEO</i>).....	32
2.3.2.2 Listing the maximal bicliques of a chordal bipartite graph	33
 CHAPTER 3 ALGORITHMS FOR FINDING THE MECB IN CONVEX	
BIPARTITE GRAPHS	36
 3.1 PROBLEM TRANSFORMATIONS	36
3.1.1 Graph to interval transformation.....	36
3.1.2 Interval to point transformation	37
 3.2 ALGORITHM DESCRIPTION	39
3.2.1 Algorithm BICLIQUE_DOM.....	39
3.2.2 Algorithm BICLIQUE_DOMA	41

CHAPTER 4 ALGORITHMS FOR FINDING THE MECB IN DOUBLY CONVEX BIPARTITE GRAPHS AND BIPARTITE PERMUTATION GRAPHS 48

4.1 Finding the MECB in doubly convex bipartite graph.....	48
4.1.1 Transforming a doubly convex bipartite graph into an orthogonally convex rectilinear polygon	49
4.1.2 Finding the MAAPR in an orthogonally convex rectilinear polygon.....	53
4.1.2.1 Two types of maximal rectangle exist in the orthogonally convex rectilinear polygon	54
4.1.2.2 The implicit area matrix M is totally monotone	55
4.1.2.3 The implicit area matrix M is a totally monotone partial matrix	57
4.1.3 Algorithm description	59
4.2 Finding the MECB in bipartite permutation graphs.....	60
4.2.1 A linear optimal time solution	61
4.2.2 Algorithm description	62

CHAPTER 5 IMPLEMENTATION AND EXPERIMENTAL RESULTS 64

5.1 Implementation	64
5.1.1 System environment.....	65
5.1.2 General design issues.....	65
5.1.3 Specific issues related to individual algorithms.....	66
5.2 Experimentation.....	69
5.2.1 Input generation	69

5.2.2 Experiment 1: Running time on convex bipartite graphs	70
5.2.3 Experiment 2: Running time on doubly convex bipartite graphs	75
CHAPTER 6 CONCLUSIONS AND DIRECTIONS FOR FUTURE WORK ...	79
6.1 Conclusions.....	79
6.1.1 Convex bipartite graphs	79
6.1.2 Doubly convex bipartite graphs and bipartite permutation graphs	81
6.1.3 Rules for choosing the best performance algorithms.....	82
6.2 Directions for future work	83
6.3 Recent developments	84
6.3.1 A bird-view of the grid.....	84
6.3.2 Algorithm BICLIQUE_DOMAL.....	89
6.3.3 Analysis of complexity	92
BIBLIOGRAPHY	95

List of figures

Figure 1. Convex bipartite graphs.....	9
Figure 2. Orderings of vertices of the bipartite permutation graph	10
Figure 3. Types of single staircase matrix.	23
Figure 4. An interval representation of bipartite graphs	37
Figure 5. Grid representation of intervals.	38
Figure 6. Graph to interval transformation.	49
Figure 7. Interval to polygon transformation.	50
Figure 8. This example illustrates the defining points	53
Figure 9. An example illustrating the totally monotonicity of the Area matrix.....	56
Figure 10. A schematic drawing of a totally monotone rising double staircase matrix...	58
Figure 11. Running time in Experiment 1.....	71
Figure 12. Normal distribution vs. uniform distribution.....	74
Figure 13. Running time in Experiment 2.....	77
Figure 14. Two exemplary orthogonally convex polygons	78
Figure 15. A surface view of a portion of a grid.....	84
Figure 16. A schematic view of the ridges.....	85
Figure 17. Evolution of CLM.	86
Figure 18. A graphical interpretation of the dynamics of CLMs.....	88
Figure 19. Weeding of insignificant peaks..	90

List of tables

Table 1. An exemplary microarray data matrix.	2
Table 2. Summary of algorithm complexities.	6
Table 3. Results on convex bipartite graphs.	71
Table 4. Results on doubly convex bipartite graphs.	76

Chapter 1 Introduction

In this study, the problem of finding the maximum edge cardinality biclique (MECB) was investigated in the context of biclustering analysis of DNA microarray data matrices. Microarray technology is one of the latest breakthroughs in experimental molecular biology. Using this revolutionary technology, biologists are able to quantitatively monitor expression patterns of thousands of genes simultaneously. Since the birth of microarray technology in 1995 [32], vast amounts of valuable gene expression data have been routinely produced. Due to the inherent complexity of biological systems, sophisticated data analysis techniques are required to discern knowledge that is hidden in large-scale gene expression data sets. As a form of unsupervised learning, clustering analysis is frequently used for this purpose.

1.1 Microarray data matrix and biclustering analysis

Microarray data are often presented as two-dimensional matrices. The matrix rows correspond to genes (or clones, open reading frames, etc.) and the matrix columns corresponds to test conditions (or samples, treatments, time points, etc.). Each entry of the matrix represents the expression level of a given gene measured under a given condition. Biologists are often interested in finding a subset of genes that are up-regulated

or down-regulated in a coherent fashion under a subset of conditions, or in finding a subset of conditions (or drugs, diseases) that consistently affect the expression of a subset of genes. Traditional clustering methods cluster either genes or conditions separately, and thus do not provide solution to these needs of biologists. Consequently, biclustering analysis techniques that aim to catch direct correlations between genes and conditions have recently become popular tools used by the bioinformatics community [8]. According to [35], “In gene expression data, a bicluster is a subset of the genes exhibiting consistent patterns over the subset of the conditions”, (See Table 1) and the process of detecting biclusters is termed biclustering analysis. The biclustering problem is generally believed to be NP-hard. Several heuristic biclustering algorithms have been reported in the literature, and most of them are based on greedy and probabilistic approaches [8,15,23,6,35,38,21]. While these algorithms do have technical merits in finding biclusters, they also have various limitations. Thus, more robust biclustering methods are still fervently pursued.

Table 1. An exemplary microarray data matrix. Entries are transformed from numerical values to signs to indicate changes of gene expression levels. ‘+’ : significant increase, ‘-’ : significant decrease, ‘ $\emptyset$ ’ : no significant change. Note that the subset of genes (G_2, G_3, G_7) and the subset of conditions (C_2, C_3, C_4, C_5) form a bicluster.

Gene \ Condition	C_1	C_2	C_3	C_4	C_5
G_1	+	-	+	+	$\emptyset$
G_2	$\emptyset$	+	+	-	+
G_3	-	+	+	-	+
G_4	+	+	-	$\emptyset$	-
G_5	-	-	+	+	+
G_6	+	$\emptyset$	-	-	+
G_7	+	+	+	-	+

1.2 The DNA Microarray data modelling using bipartite graph

A microarray data matrix can be modeled as a graph $G = (V, E)$. A vertex $v \in V$ of the graph represents either a gene or a condition that is tested, and an edge $e \in E$ exists if and only if the expression level of the given gene increases significantly under the given condition. The resulting graph is a bipartite graph [8,35]. Frequently, biclustering analysis involves finding the maximum bicluster in the expression data. The maximum bicluster is the bicluster whose product of number of genes and number of conditions is the maximum among all biclusters. Under the bipartite graph model, a bicluster will be represented as a complete bipartite subgraph, also known as biclique. The problem of finding the maximum bicluster is now reduced to the problem of finding the maximum edge cardinality biclique (MECB), in bipartite graphs.

Unfortunately, the problem of finding maximum edge cardinality bicliques in bipartite graphs and related decision problems have also been shown to be NP-complete [11,28] and hence only approximation algorithms [18] and output sensitive polynomial time algorithms [3] have been reported. However, we have noticed that for certain classes of bipartite graphs, such as chordal bipartite graphs, domino-free bipartite graphs and convex bipartite graphs, polynomial-time algorithms do exist for the problem of listing maximal bicliques in a given graph [3,20,4]. In this study, we show how to find MECB efficiently in convex bipartite graphs.

Convex bipartite graphs were introduced by Glover [16]. For this class of graphs, Alexe et al. [3] recently suggest that the problem of finding MECB is polynomially solvable.

For a graph with N vertices, the variant of their output sensitive *consensus algorithms* named *modular input consensus algorithm* has an upper bound of $O(N^3\beta)$ in time complexity and uses $O(|E|)$ storage space, where β is the number of maximal bicliques to be output by the algorithm. For a B -convex bipartite graph $G = (A, B, E)$, $N = |A| + |B|$ and $\beta = O(|B|^2)$ [3]. Thus, at least $O((|A| + |B|)^3 |B|^2)$ time and $O(|E|)$ space are required. Alternatively, we could use algorithms for listing maximal bicliques in chordal bipartite graphs to find the MECB [20], as convex bipartite graphs are a subclass of chordal bipartite graphs [5]. In this case, the time and space complexities are $O(|A| + |B| + |E|)$ and $O(|E|)$, respectively. We argue that substantial improvements in time and space complexities can be achieved by taking advantage of the structural property of convex bipartite graphs that the neighborhood of each vertex $a \in A$ is an interval.

1.3 Motivation of the study

The motivation for studying convex bipartite graphs in the context of biclustering of gene expression data is that linear ordering of genes exists naturally in several forms. (Such orderings include chronological ordering in the course of evolution, spatial ordering on chromosomes.) Given a bipartite graph, Booth and Lueker's algorithm [7] can be used to test for convexity in time linear in the number of edges. This algorithm also produces a convex ordering, if one exists. In case that the gene expression data can be modeled as a convex bipartite graph, our algorithm can be used directly to find the MECB more efficiently. Even if the underlying bipartite graph is not convex, we may be able to obtain subgraphs that are convex using Booth and Lueker's algorithm mentioned above. Upon identifying the MECB in the subgraphs, the MECB in the bipartite graph may then be found by extending the results from the subgraphs in time linear in the number of edges

[3]. Therefore, the MECB finding algorithm may serve as a core component in a large bicluster analysis system.

In addition to biclustering analysis of DNA microarray data matrix, which is a form of data mining [8,35], both MECB and convex bipartite graph have interesting applications in industrial optimization problems and thus received extensive consideration [16,11,24].

1.4 Contributions of the study

Our main contributions are: (i) Two algorithms for the MECB problem in convex bipartite graphs; (ii) An almost linear time algorithm for the problem of finding the MECB in doubly convex bipartite graphs; (iii) A linear time algorithm for the MECB problem in bipartite permutation graphs. The time and space complexities of these algorithms are given in Table 2. Compared to the existing algorithm by Kloks et al. [20], named CHORDAL in our study, the improvement in space requirement is significant, as $|E| = O(|A||B|)$. The improvement in time complexity depends on $|A|$ and $|B|$. In case where $|A| \leq |B|$, the algorithm BICLIQUE_DOMA represents a significant improvement over the algorithm CHORDAL; if $|B| < |A|$, the algorithm BICLIQUE_DOM will perform better than the algorithm CHORDAL. In fact, our experimental results (See Chapter 5) demonstrate that the space requirement plays a vital role in determine the running time of these algorithms when the size of the graph is large. For doubly convex bipartite graphs and bipartite permutation graphs, the algorithms BICLIQUE_MAAPR and BICLIQUE_PERM are definitely superior to the algorithm CHORDAL in terms of both time and space complexity.

Table 2. Summary of algorithm complexities. Note: $d_{\max}(B)$ is the maximum degree of B . The symbol $\alpha(\bullet)$ denotes the inverse of the Ackermann's function.

Algorithm	Applicable graph type	Time	Space
BICLIQUE_DOMA	B -Convex bipartite	$O(d_{\max}(B) A)$	$O(A)$
BICLIQUE_DOM	B -Convex bipartite	$O(B ^2 + A)$	$O(B + A)$
BICLIQUE_MAAPR	Doubly convex bipartite	$O(A \alpha(A))$	$O(A)$
BICLIQUE_PERM	Bipartite permutation	$O(A)$	$O(A)$
CHORDAL(by Kloks et al.)	Chordal bipartite	$O(A + B + E)$	$O(E)$

1.5 Thesis organization

The remainder of the thesis is organized as follows: Chapter 2 provides graph theoretical preliminaries, previous results on related problems, and solutions and algorithms that will be used as components in our algorithms. The major theoretical contributions of this study are given in Chapter 3 and 4. In Chapter 3, two algorithms are presented which solve the MECB problem in convex bipartite graphs. Chapter 4 presents optimal solutions to the problem of finding the *MECB* in doubly convex bipartite graphs and bipartite permutation graphs. Chapter 5 presents experimental results of our algorithms and compares the results to related algorithms of other investigators. Chapter 6 summaries this research, presents directions for future work and conclusions.

Chapter 2 Foundation of this study

This chapter presents definitions, notations and basic algorithms that are used in this thesis. Section 2.1 provides graph theoretical definitions and notations. Section 2.2 introduces existing results on layered range tree and matrix searching algorithms, which will be utilized in our algorithms as subroutines. In Section 2.3, related work on listing maximal bicliques of a given graph will be discussed to depict the frontier on this line of researches.

2.1 Preliminaries

Standard graph theoretic notations and definitions used in contemporary texts [12,5] are followed. Throughout the study, all graphs $G = (V, E)$ are assumed to be finite, simple, and undirected, unless noted otherwise. A graph $G = (V, E)$ consists of a finite set V of vertices and a collection E of edges, where each edge $e \in E$ is a 2-element subsets of V , i.e., $e = (\{v_i, v_j\} \mid v_i \neq v_j \text{ and } v_i, v_j \in V)$. A sequence of distinct vertices $v_1, \dots, v_k$ is a *path*, if $(v_i, v_{i+1}) \in E$ for each $1 \leq i \leq k - 1$. The length of the path is the number of edges in the path. A *cycle* is a path of length at least 3, beginning and ending with the same vertex ($v_1 = v_k$). An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of the cycle. The neighbourhood of a vertex $v \in V$ is the set $\lambda(v) = \{u \in V \mid (v, u)$

$\in E\}$. The degree of a vertex v , denoted by $d(v)$, is the number of vertices in $\lambda(v)$. Given a graph $G = (V, E)$ and a subset $H \subseteq V$, the subgraph induced by H is the graph $G[H] = (H, F)$, where $F = \{(u, v) \mid (u, v) \in E, u, v \in H\}$. A graph $G = (V, E)$ is a **bipartite graph**, or two-colourable, if V can be partitioned into two non-empty vertex sets $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_m\}$ such that $A \cup B = V$, $A \cap B = \emptyset$, and for every edge $(a, b) \in E$, where $E \neq \emptyset$, $a \in A$, $b \in B$. A bipartite graph can be represented using a *bipartite adjacency matrix* which is an $|A| \times |B|$ $(0, 1)$ -matrix M with $M_{ij} = 1$ if and only if $(a_i, b_j) \in E$.

An ordering π of a set of vertices $V = \{v_1, \dots, v_n\}$ is a bijection $\pi: V \rightarrow \{1, \dots, n\}$ such that $\pi(v_i) = i$ for $1 \leq i \leq n$. A bipartite graph $G = (A, B, E)$, with $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_m)$, is *convex on B* if there exists an ordering, π^B , of the vertices in B , such that for every vertex $a \in A$, if

$$(a, b_i) \in E, (a, b_j) \in E, (1 \leq i < j \leq m), \text{ then}$$

$$(a, b_k) \in E, (i \leq k \leq j).$$

Here, π^B is said to have the *convex property*. In other words, for each vertex $a \in A$, $\lambda(a) = \{b_i, \dots, b_j\}$ consists of vertices which are consecutive in π^B . Consequently, in π^B , $\lambda(a) = (b_i, \dots, b_j) = [i, j] = I_a$ is an interval on the integer domain $[1, m]$, where i and j are endpoints of the interval I_a . We call i left endpoint and j right endpoint of I_a respectively. Similarly, we call b_i starting vertex and b_j finishing vertex of $\lambda(a)$. The term *convex on A* is analogously defined. A **convex bipartite graph** is a bipartite graph that is convex on one of the vertex sets. In the remainder of the thesis, unless noted otherwise, all convex

bipartite graphs are B -convex bipartite graphs (i.e., if only one of the vertex sets of the bipartite graph has a convex ordering, we call this set B , the other set A), and vertices in B are labelled with integers in the order of π^B . Thus, for an ordered set of vertices $\{b_i, \dots, b_j\}$ of B that constitute the neighbourhood of a vertex $a \in A$, i.e., $\{b_i, \dots, b_j\} = \lambda(a)$ and $\{b_i, \dots, b_j\}$ are labelled with consecutive integers from the integer domain $[1, m]$, the notation $\lambda(a)$, $\{b_i, \dots, b_j\}$, $[i, j]$ and I_a are used interchangeably. An example of B -convex bipartite graph is shown in Figure 1.a.

Figure 1. Convex bipartite graphs. **a)** A B -convex bipartite graph. **b)** A doubly convex bipartite graph. **c)** A bipartite permutation graph. The shaded vertices in each graph represent the maximum edge cardinality bicliques.

A bipartite graph $G = (A, B, E)$ is called a **doubly convex bipartite graph** (also known as biconvex graph) if it is convex on both A and B (Figure 1.b). A subclass of the doubly convex bipartite graphs is the class of **bipartite permutation graphs**. A graph is a **permutation graph** if there is a pair of orderings of the vertex set such that there is an edge between vertices a and b if and only if a precedes b in one of the orderings, while b precedes a in the other [34]. A bipartite permutation graph is a permutation graph whose vertex set is two-colourable (Figure 1.c).

The two orderings underlying the bipartite permutation graph shown in Figure 1.c are giving in Figure 2.

Figure 2. Orderings of vertices of the bipartite permutation graph shown in Figure 1.c.

Given a bipartite graph $G = (A, B, E)$, a **biclique** $K_{p,q}$ is a complete bipartite subgraph of G , i.e. $K_{p,q} = (X, Y)$ such that $X \subseteq A$, $Y \subseteq B$, $|X| = p$, $|Y| = q$, and for every vertex $x \in X$, $y \in Y$, the edge $(x, y) \in E$. Some researchers [31] define bicliques as inclusion-maximal induced complete bipartite subgraphs, which is equivalent to the maximal bicliques defined below. We will follow the first definition, which is more general and includes both maximal and non-maximal bicliques. If G is a bipartite graph, $K_{p,q}$ is an induced subgraph of G ; otherwise, $K_{p,q}$ is not necessarily an induced subgraph of G . A biclique $K_{p,q} = (X, Y)$ is a **maximal biclique** if there exists no other biclique $K'_{p,q} = (X', Y')$ in G such that $X \subseteq X'$ and $Y \subseteq Y'$. Apparently, a biclique $K_{p,q}$ has $p + q$ vertices and $p * q$ edges. The maximum biclique can either be a maximum vertex biclique that maximizes $(p + q)$ or a maximum edge biclique that maximizes $(p * q)$. The maximum edge biclique is the subject of this study. A biclique $K_{p,q}$ is the **maximum edge cardinality biclique (MECB)** in G if the product of p and q is the maximum among all bicliques in G . The term “edge cardinality” is used to further indicate that only non-weighted graphs are concerned here. We will drop the subscripts p, q whenever the context permits.

2.2 Existing results

Results in the following areas, including layered range trees, totally monotone matrices and totally monotone partial matrices will be used as sub-procedures in our algorithms. Thus, a detailed account of background knowledge in these areas is warranted.

2.2.1 Layered range tree related algorithms

As pointed out in Section 2.1, for a B -convex bipartite graph $G = (A, B, E)$, the neighbourhood of each vertex $a \in A$, is an integer interval I_a on the domain $[1, m]$. We will see that one of our algorithms relies heavily on efficient interval enclosure queries. Let $I_1 = [i_1, j_1]$ and $I_2 = [i_2, j_2]$ be a pair of intervals. We say I_1 encloses I_2 , denoted by $I_2 \subseteq I_1$, if $i_1 \leq i_2$ and $j_1 \geq j_2$. An interval enclosure query answers the following question: Given a set of intervals A and a query interval $I_q = [i, j]$, find the subset of intervals $S_I = \{[p, q] \in A: p \leq i \text{ and } q \geq j\}$. Given a pair of 2-dimensional points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we say p_1 dominates p_2 , denoted by $p_1 \succeq p_2$, if $x_1 \geq x_2$ and $y_1 \geq y_2$. The interval enclosure query has been shown to be equivalent to the point dominance query in a 2-dimensional space [17]. Indeed, given two intervals $I_1 = [i_1, j_1]$ and $I_2 = [i_2, j_2]$, if we transform them into 2-dimensional points $p_1 = (-i_1, j_1)$ and $p_2 = (-i_2, j_2)$, clearly, $p_1 \succeq p_2$ is equivalent to that $I_2 \subseteq I_1$. The 2-dimensional point dominance query can be answered efficiently using the so-called layered range tree [30] due to Lueker [26] and Willard [37].

Theorem [30]: *The layered range tree built on a set of n 2-dimensional points requires $O(n \log n)$ storage and can be constructed in time $O(n \log n)$. Using the layered range tree, we can find the dominance number of a given point in $O(\log n)$ time.*

The following is a brief description of the *layered range tree* data structure, which is sometimes called *range tree with fractional cascading*. Although the layered range tree data structure can handle data points in arbitrarily high dimensions, we will only talk about it in 2-dimensional space in our particular case.

Given a sequence of real numbers $S = \{x_1, x_2, \dots, x_{n-1}, x_n\}$ in increasing order, a segment tree on S is a complete binary search tree whose leaves contains the intervals $[x_1, x_2)$, $[x_2, x_3)$, ..., $[x_{n-1}, x_n)$, $[x_n, x_n]$, and an internal node v corresponds to the interval $[x_i, x_j)$ spanned by the union of intervals in the subtree rooted at v . A layered range tree T for a set of 2-dimensional points consists of a primary segment tree on the x -coordinates. The root node of T contains a pointer to a balanced binary search tree built on the y -coordinates of all points in T . Each internal node v of T (except the root) contains a pointer to a y -list containing points whose x -coordinates project to the interval of v , and the y -list is sorted by y -coordinates in decreasing order. Meanwhile, associated with each element y_i of the y -list are two downward pointers pointing to the positions of the smallest element that is larger than or equal to y_i in the y -lists of the children of v .

2.2.1.1 Construction of a layered range tree

From the above description, we know that each node v is associated with a y -list. In case v is the root, $y\text{-list}(v)$ points to a balanced binary search tree $y\text{-tree}$. Therefore, inside a

layered range tree T , there are pointers to two trees: $Primary(T)$ points to a primary segment tree built on distinct x -coordinates, and $Auxiliary(T)$ points to a balanced binary search tree built on distinct y -coordinates.

In case the x -coordinates are not distinct, points with the same x -coordinates can be grouped together into an aggregated structure which contains x as a key and a list of y -coordinates. All y -coordinates in the list are associated with the x , and are sorted in descending order. We can sort all points first by x -coordinates in ascending order and then y -coordinates in descending order before construction of the layered range tree; namely, for a pair of points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, if $x_1 < x_2$, then p_1 precedes p_2 regardless of y_1 and y_2 ; in case $x_1 = x_2$, p_1 precedes p_2 if $y_1 > y_2$.

A node v of the primary tree has the following fields:

$LEFT(v)$: left child of v

$RIGHT(v)$: right child of v

$KEY(v)$: key of v , it is the x_{mid} (median of x -coordinates) of all points contained in v

$y-list(v)$: a pointer to a $y-list$. Each element w in the $y-list$ has 6 fields: $key(w)$, $rank(w)$, $prev(w)$, $next(w)$, $left_pointer(w)$ and $right_pointer(w)$ as defined above.

A node v of the auxiliary tree has the following fields:

$LEFT(v)$: left child of v

$RIGHT(v)$: right child of v

$KEY(v)$: key of v , it is the y_{mid} (median of y -coordinates) of all points contained in v

$A_NODE(v)$: a pointer to an element w in the y -list on which the threaded binary search tree is built.

An algorithm for the construction of the layered range tree is given below. The algorithm for building and querying the auxiliary tree is omitted here, as it is straightforward.

ALGORITHM Build_layered_range_tree (L)

Input: A list L of 2-dimensional points sorted in increasing order in x -coordinates.

Output: The root of a layered range tree for L .

Primary(T) = Build_primary_tree(L)

Let L_y be the y -list associated with the root node of the primary tree, then

Auxiliary (T) = Build_balanced_binary_search_tree(L_y)

Return T

END Build_layered_range_tree

ALGORITHM Build_primary_tree (L)

Input: A list L of 2-dimensional points sorted in increasing order in x -coordinates.

Output: The root of a primary binary search tree for L .

$n \leftarrow$ the number of elements in L (assume all x -coordinates are distinct)

IF $n = 1$

THEN make a leaf node v ,

y -list(v) $\leftarrow L$

RETURN v .

```

 $m \leftarrow \lfloor n/2 \rfloor$ 

 $x_{mid} \leftarrow L[m].x$ 

 $LL \leftarrow L[1] \text{ to } L[m]$ 

 $RL \leftarrow L[m+1] \text{ to } L[n]$ 

 $v \leftarrow \text{a new node}$ 

 $KEY(v) \leftarrow x_{mid}$ 

 $LEFT(v) \leftarrow Build(LL)$ 

 $RIGHT(v) \leftarrow Build(RL)$ 

Merge the  $y\text{-list}(LEFT(v))$  and  $y\text{-list}(RIGHT(v))$  into one list  $L_{merge}$  sorted in
descending order. For each element  $w$  in  $L_{merge}$ , if it is from  $LEFT(v)$ , then
 $left\_pointer(w) \leftarrow$  corresponding element in  $y\text{-list}(LEFT(v))$ ,  $right\_pointer(w) \leftarrow$  last
merged element from  $y\text{-list}(RIGHT(v))$ , vice versa.

 $y\text{-list}(v) \leftarrow L_{merge}$ 

RETURN  $v$ 

END Build_primary_tree

```

2.2.1.2 Dominance counting query on a layered range tree

Ideas for answering dominance counting queries using layered range trees were briefly mentioned in [30], but a detailed description is lacking. Thus, a concise description of the dominance counting query algorithm outlined in [30] is given in the remainder of this section. The layered range tree can also be used to answer reporting queries. But in this study, reporting query is not relevant. Hence, algorithms for reporting query will not be reviewed here.

Let $p = (x_p, y_p)$ be a 2-dimensional point, and T be a layered range tree. For a dominance counting query, we first perform an initial search on the auxiliary tree to locate the node u with the smallest $key(u)$ such that $key(u) \geq y_p$. Recall that $A_NODE(u)$ has two downward pointers, $left_pointer$ and $right_pointer$ that tell the position of the smallest y that is greater than or equal to y_p in the y -lists of the children of $root(T)$. From now on, we only need to follow these pointers to find the rank of y_p in the subsequent tour of the primary tree T . For an internal node v of T , let w be the smallest element in the $y_list(v)$ such that $key(w) \geq y_p$, if $key(v) < x_p$, add the rank of the element pointed by $left_pointer(w)$ to the count, and continue with the right child of v ; otherwise, go to the left child of v . If v is a leaf, add the rank of the smallest element w in the $y_list(v)$ such that $key(w) \geq y_p$.

ALGORITHM Counting_query (T, p)

Input: a layered range tree T , and a query point $p = (x_p, y_p)$.

Output: COUNT = The number of points that dominate p .

$a_node \leftarrow \text{Query_y_tree}(\text{Auxiliary}(T), y_p)$ /* Locate the smallest element a_node in the auxiliary tree such that $key(a_node) \geq y_p$. */

COUNT $\leftarrow \text{Counting}(\text{Primary}(T), x_p, a_node)$.

Return COUNT

END Counting_query

ALGORITHM Counting(v, x, a_node)

Input: a node v of the primary tree, an x value and a pointer that identifies the element in the $y\text{-list}(v)$ to be monitored

Output: Count = The number of points that are dominated by p in the subtree rooted at v

If v is a leaf and $x \geq \text{key}(v)$, return $\text{rank}(a_node)$

If v is a leaf and $x < \text{key}(v)$, return 0

Left_node = left_pointer(a_node)

Right_node = right_pointer(a_node)

If $x = \text{KEY}(v)$

$Count \leftarrow \text{Rank}(\text{Left_node})$

Else If $x > \text{KEY}(v)$

$Count \leftarrow \text{Rank}(\text{Left_node}) + \text{Counting}(\text{RIGHT}(v), x, \text{Right_node})$

Else

$Count \leftarrow \text{Counting}(\text{LEFT}(v), x, \text{Left_node})$

Return $Count$

END Counting

2.2.2 Totally monotone matrix and the SMAWK algorithm

Our algorithm for finding the MECB in bipartite permutation graphs involves searching of a totally monotone matrix for the maximum entry. Thus, a brief introduction to the concept of totally monotone matrix and the searching method is first provided.

A $p \times q$ matrix $M = (M_{ij})$ with real entries is said to be *monotone* if for $1 \leq i < k \leq p$, we always have $j(i) \leq j(k)$, where $j(i)$ denotes the index of the leftmost column such that $M_{i,j(i)}$

is equal to the maximum entry in row i (so does $j(k)$). M is *totally monotone* if every submatrix of M is monotone; in other words, for every i, j, k, l such that $i < k, j < l, M_{ij} \leq M_{il}$ implies $M_{kj} \leq M_{kl}$.

In [2], it is shown that given a $p \times q$ monotone (but not necessarily totally monotone) matrix M , the maximum entries in all rows of M can be found in time $O(q \log p)$ with a straightforward divide-and-conquer algorithm. This is not difficult to explain. Let $i = \lceil p/2 \rceil$ then, $j = j(i)$ can be found in time $O(q)$. Now, divide M into four submatrices as follows: $M_1 = M[1, \dots, i-1; 1, \dots, j]$, $M_2 = M[1, \dots, i-1; j+1, \dots, q]$, $M_3 = M[i+1, \dots, p; 1, \dots, j-1]$, and $M_4 = M[i+1, \dots, p; j, \dots, q]$.

Since M is monotone, it is impossible for the row maxima to fall in the submatrices M_2 and M_3 . To find the maxima for rows other than row i , we only need to recursively find the row maxima on the submatrices M_1 and M_4 . The following recurrence gives the time required by this algorithm:

$$f(p, q) \leq q + \max_{1 \leq j \leq q} (f(\lceil n/2 \rceil - 1, j) + f(\lfloor n/2 \rfloor, q - j + 1))$$

with $f(0, q) = f(p, 1) = \text{constant}$. Solving this recurrence, we have $f(p, q) = O(q \log p)$. Aggarwal et al. [2] also proved that $\Omega(q \log p)$ is a lower bound for the problem of finding row maxima in a monotone matrix.

However, if the additional property of total monotonicity is present for matrix M , a better upper bound is achievable; in fact, a linear time algorithm, known as SMAWK, was provided by Aggarwal et al. [2]. The basic idea of the algorithm is to exclude as many

columns as possible from further considerations by taking advantage of the total monotonicity. In particular, for any three columns $a < b < c$, if there is a row r such that $M_{rb} \leq M_{ra}$, and $M_{rb} \leq M_{rc}$, then the column b can be safely excluded; the reason for this is: let $1 \leq i \leq r$, $M_{rb} \leq M_{ra}$ implies $M_{ib} \leq M_{ia}$ given $M[i, r; a, b]$ is monotone; similarly, let $r \leq k \leq p$, $M_{rb} \leq M_{rc}$ implies $M_{kb} \leq M_{kc}$ given $M[r, k; b, c]$ is monotone. The original SMAWK algorithm treated the cases where $q \geq p$ and $q < p$ differently. In the case of $q \geq p$, a subroutine called *REDUCE* is used to reduce the number of columns to p (for any $p \times q$ matrices, regardless the number of columns q , we only need at most p columns to contain all row maxima, at the extreme situation, all row maxima can be contained in just one column).

ALGORITHM REDUCE(M)

$C \leftarrow M, k \leftarrow 1.$

WHILE C has more than p columns DO

CASE

$C(k, k) \geq C(k, k + 1)$ and $k < p$: $k \leftarrow k + 1.$

$C(k, k) \geq C(k, k + 1)$ and $k = p$: delete column $k + 1.$

$C(k, k) < C(k, k + 1)$: delete column k ; IF $k > 1, k \leftarrow k - 1.$

END CASE

RETURN (C)

The main algorithm MAXCOMPUTE is recursive. The underlying idea is to find the position (column index) of row maxima for even-numbered rows recursively, then from

the known position of the even rows, find the maxima in odd rows by exploiting the total monotonicity.

ALGORITHM MAXCOMPUTE (M)

1. $B \leftarrow \text{REDUCE } (M)$.
2. IF $p = 1$, output the maximum and return.
3. $C \leftarrow B[2, 4, \dots, 2\lfloor p/2 \rfloor, 1, 2, \dots, q]$.
4. MAXCOMPUTE (C).
5. From the known positions of the maxima in the even rows of B , find the maxima in its odd rows

Theorem 4.3[2]: *When $p \leq q$, MAXCOMPUTE solves the maximum problem on a totally monotone $p \times q$ matrix in time $O(p + q) = O(q)$.*

In the case where $p > q$, a $\theta(q(1 + \log(p/q)))$ algorithm was proposed, which employed a combination of the MAXCOMPUTE algorithm and the divide-and-conquer algorithm for monotone matrices briefly sketched previously. Specifically, let M be a $p \times q$ totally monotone matrix with $p > q$. First, make a $q \times q$ submatrix M' out of M in such a way that row i of M' , denoted by $r_i = \lfloor i \cdot p/q \rfloor$, for $1 \leq i \leq q$. Then the MAXCOMPUTE can be applied to M' in $O(q)$ time to get $j_1, j_2, \dots, j_q$ where $j_i = j(r_i)$. Recall that $j(r_i)$ denotes the leftmost maximum entry in row r_i . The next step is to apply the divide-and-conquer algorithm to the q submatrices $S_i = M[r_{i-1} + 1, r_{i-1} + 2, \dots, r_i - 1; j_{i-1}, j_{i-1} + 1, \dots, j_i]$ ($1 \leq i \leq q$) interspersed between the rows and columns of M' . The number of rows for each S_i is at most

$\lfloor p/q \rfloor$, and, by the total monotonicity of M , the number of columns of S_i is $j_i - j_{i-1} + 1$. So, it takes time $O((j_i - j_{i-1} + 1) \log(p/q))$ to find the maxima in S_i . Some over all i , the total time for this step is $O(q \log(p/q))$. Altogether, the time for the entire algorithm is $O(q(1 + \log(p/q)))$. Since it is also proved that $\Omega(q(1 + \log(p/q)))$ is a lower bound for the case where $p > q$, this is an optimal time algorithm.

As pointed out by Daniels et al. [10], a unified upper bound for both of the cases (when $p \leq q$ and when $p > q$) would be $O(p + q)$, as $O(q(1 + \log(p/q))) = O(q(1 + p/q)) = O(p + q)$. In fact, the MAXCOMPUTE algorithm can be applied to the second case as well if the REDUCE algorithm is allowed to take on totally monotone matrices with $p > q$ and to remove as many columns as possible. Using an auxiliary stack data structure, this can be easily achieved as evidenced by a Python implementation of the SMAWK algorithm [14]. This approach is also used later in our implementation of the REDUCE procedure.

2.2.3 Totally monotone partial matrices and related searching algorithms

Later on in Chapter 4, we will show that the problem of finding the MECB in doubly convex bipartite graph can be transformed to the problem of finding the maximum area axis-parallel rectangle (MAAPR) in an orthogonally convex rectilinear polygon. The best-known algorithm for the latter problem involves searching for a maximum in totally monotone partial matrices, which is going to be reviewed here.

A sequence of integers is said to be unimodal if there is only one peak or one trough in the entire sequence. In [1], the concept of totally monotone partial matrices was

introduced as a generalization of the totally monotone matrix. A $p \times q$ matrix $M = (M_{ij})$ whose entries are either real numbers or blanks is called a ***totally monotone partial matrix*** if there are two unimodal sequences $\{s_i\} = s_1, \dots, s_p$ and $\{f_i\} = f_1, \dots, f_p$ of integers between 0 and $q + 1$ with the following properties:

- (a) For each i we have $s_i \leq f_i$.
- (b) M_{ij} is a real number if and only if $1 \leq s_i \leq j \leq f_i \leq q$. Otherwise M_{ij} is blank.
- (c) If $i < k$, $j < l$, M_{ij} , M_{il} , M_{kj} , and M_{kl} are real, $M_{ij} \leq M_{il}$ implies $M_{kj} \leq M_{kl}$.

For $1 \leq i \leq p$ $\{s_i\}$ and $\{f_i\}$ are also known as boundary sequences of M , i.e., s_i and f_i are the column indices of the first and last non-blank entries in row i , respectively.

The maximum entry in a $p \times q$ totally monotone partial matrix M cannot be found directly using the SMAWK algorithm due to the presence of illegal (blank) entries. However, M can be partitioned into a set of disjoint special submatrices called totally monotone single staircase matrices whose row maxima can be found in almost linear time using existing algorithm. It should be mentioned that the partitioning step could be done in linear time, i.e., $O(p + q)$. Four types of totally monotone single staircase matrix can be defined using $\{s_i\}$ and $\{f_i\}$ [1]:

- (A) *Rising staircase*: $\{s_i\}$ contains all 1's and $\{f_i\}$ is a non-increasing sequence;
- (B) *Falling staircase*: $\{s_i\}$ is a non-decreasing sequence and $\{f_i\}$ contains all q 's;
- (C) *Reverse falling staircase*: $\{s_i\}$ contains all 1's and $\{f_i\}$ is a non-decreasing sequence;
- (D) *Reverse rising staircase*: $\{s_i\}$ is a non-increasing sequence and $\{f_i\}$ contains all q 's.

Schematic examples for these four types are shown in Figure 3. Note that in [19], type B was called reverse falling staircase and Type C was called falling staircase, in this paper, we adopt the terminology of [1]. It is also noteworthy that a totally monotone single staircase matrix of one type can be converted into another totally monotone single staircase matrix of another type through simple operations. For example, as shown in Figure 3, a totally monotone rising staircase matrix can be converted into:

- (a) a totally monotone falling staircase matrix by negating all non-blank entries and reversing the order of columns, or
- (b) a totally monotone reverse falling staircase matrix by negating all non-blank entries and reversing the order of rows.

Figure 3. Types of single staircase matrix. A: Rising staircase. B: Falling staircase. C: Reverse falling staircase. D: Reverse rising staircase. Refer to text for details.

The implication of this convertibility is that if we have an algorithm for finding the row maxima in totally monotone rising staircase matrices, then it can be used to find the row minima in totally monotone falling or reverse falling staircase matrices, and vice versa.

For falling and reverse falling staircase matrices, row maxima can be found in linear time using the SMAWK algorithm [2] after completing the blank entries according to certain rules that preserve the total monotonicity (See *Lemma 3.3* of [1], for some examples, see [19]). But totally monotone rising and reverse rising staircase matrices cannot be completed without introducing new row maxima. Analogously, falling and reverse falling staircase matrices cannot be completed without introducing new row minima. The most efficient algorithm for finding row minima in these types of $p \times q$ staircase matrix is an $O(\alpha(p)q + p)$ time algorithm reported in [19], where $\alpha(p)$ is a very slowly growing function called the inverse of Ackermann's function defined in [19]. However, this algorithm has a large constant and is difficult to implement. An alternative algorithm (called `PartialMaxCompute` in the present study) by Aggarwal et al. [1] with $O((p+q) \log \log p)$ time complexity is more popular. Let us now illustrate it on a $p \times q$ totally monotone rising staircase matrix M .

The basic idea here is similar to that of the SMAWK algorithm, i.e., first find the row maxima in a subset of rows, then based on the position of these maxima, exclude substantial portions of the matrix from further consideration by exploiting the total monotonicity of the matrix. The starting point of the algorithm is the following lemma:

Lemma 3.4 of [1]: *There is a constant c_1 such that if M is a $p \times q$ totally monotone rising staircase matrix, then its row maxima can be found in $c_1(p^2 + q)$ comparisons.*

This is easy to see because the nonblank entries of a rising staircase matrix can be trivially partitioned into k column-wise disjoint totally monotone matrices $M_1, \dots, M_k$ ($k \leq p$). Let the dimension of each M_i be $p_i \times q_i$ for $1 \leq i \leq k$, using the SMAWK algorithm, row maxima of each M_i can be find in time $c_0(p_i + q_i)$, where c_0 is a constant. In this way, we can find the row maxima of all M_i in time $c_0(kp + q)$ as each p_i is at most p and the summation of q_i for $1 \leq i \leq k$ is q (recall that M_i 's are column-wise disjoint). Now we have k maxima for each row, and the true maxima of each row can be found in k comparisons with brutal force. Thus, with at most $k \cdot p \leq p^2$ additional time we can find the row maxima in the entire M . Altogether, this is $O(p^2 + q)$ time. The point here is that, if we have a $\sqrt{p} \times q$ rising staircase matrix, its row maxima can be found in time $c_0(p + q) = O(p + q)$.

Algorithm PartialMaxCompute is recursive and use SMAWK as a subroutine. The base case is if $p \leq 4$, then PartialMaxCompute finds the row maxima in the matrix with brutal force (requires $4 \cdot q$ comparisons). Otherwise, let $\{f_i\}$ be the finishing boundary sequence of M . Let $s = \lfloor \sqrt{p} \rfloor$, let k be largest integer satisfying $(k + 1) \cdot s + 1 \leq n$ and let M' be the submatrix consisting of truncated rows R_i for $1 \leq i \leq k + 1$, where R_i is the row numbered $s \cdot (i - 1) + 1$ of M , for $1 \leq i \leq k$, entries in columns after $f_{s \cdot i + 1}$ are truncated. Now M' is a s

$\times q$ rising staircase matrix, whose row maxima can be found in time $O(s^2 + q) = O(p + q)$. The remaining parts of M lying between rows of M' form horizontal bands B_i for $1 \leq i \leq k + 1$. For $1 \leq i \leq k$, B_i contains all rows numbered between R_i and R_{i+1} (exclusive). The last band B_{k+1} , if it exists, occupies at most s rows numbered $R_s + 1, R_s + 2, \dots, p$. Each B_i can be further divided into two sub-regions D_i and E_i , where $D_i = M[s*(i-1)+2, \dots, s*i; 1, \dots, f_{s*i+1}]$, $E_i = M[s*(i-1)+1, \dots, s*i; f_{s*i+1}+1, \dots, f_{s*(i-1)+1}]$. Recall that M is a rising staircase matrix; this means that $\{f_i\}$ is a non-increasing sequence. Consequently, each E_i and B_{k+1} are smaller rising staircase matrix with $s - 1$ rows, whose row maxima can be found recursively in $c(s + J_i) \log \log s$, where $J_i = f_{s*(i-1)+1} - f_{s*i+1} + 1$, for $1 \leq i \leq k + 1$. Since the summation of all columns of all E_i and B_{k+1} is q , and $s^2 \leq p$, finding row maxima in all E_i and B_{k+1} requires $c(p + q) \log \log s$ comparisons.

It follows from total monotonicity, these positions of the row maxima in $R_1, \dots, R_{k+1}$ of M' allow us to delete large portions of M that can not possibly contain row maxima. These deletions will occur in the regions D_i . Let $x_i = j(R_i)$ and $y_i = f_{s*i+1}$ for $2 \leq i \leq k$, where $j(R_i)$ is the column number of the row maximum in R_i , the following two sub-regions of M can be safely excluded:

$$L_i = M[s*(i-1)+1, \dots, s*i; 1, \dots, x_i-1]$$

$$U_i = M[1, \dots, s*(i-1)-1; x_i+1, \dots, y_i]$$

If D_i contains the row maxima, it can be verified that they can only exist in the submatrix $A_i = M[s*(i-1)+1, \dots, s*i; x_i, \dots, f_{s*i+1}]$ with the regions falling in U_v deleted, for $i+1 \leq v$

$\leq k$. Each A_i is a totally monotone matrix with $s-1$ rows and J_i columns, thus its row maxima can be found with the SMAWK algorithm in time $O(s-1 + J_i)$. Obviously, the number of overlapping columns for all A_i is at most s ; therefore, given $s^2 \leq p$, the time required for finding the row maxima in all A_i is $c_1(p - s + q + s) = O(p + q)$. Given row maxima in A_i and E_i , the row maxima in B_i can be found with $s-1$ comparisons; this implies p comparisons in total for all B_i .

In summary, the total time complexity of PartialMaxCompute is $c_0(p + q) + c_1(p + q) + c(p + q) \log \log s + p = O(p + q) \log \log p$.

2.3 Related work

The complexity of the MECB problem has been studied recently by several investigators [11,28,31]. Through reducing the 3-SAT problem to the MECB problem, Peeters [28] demonstrated that the MECB problem is NP-complete. In [11], it is shown that the weighted version of this problem and a related decision problem are also NP-complete. As for the problem of finding the MECB in bipartite graphs and general graphs (specifically, non-bipartite graphs), a brutal force search is required as the number of maximal bicliques in these graphs are exponential in the number of vertices [31,13]. Or an approximation can be found by formulating the MECB problem as a linear programming problem, where the objective is to minimize the total number of edges deleted so that the remaining subgraph contains a biclique; as some edges of the remaining subgraph do not participate in the formation of the biclique, a remaining subgraph with maximum edge cardinality does not necessarily contains the MECB [18]. A closely related problem is to enumerate all maximal bicliques in a given graph. As the

MECB must be one of the maximal bicliques, the MECB problem can be solved trivially with the same time complexity upon listing all maximal bicliques. However, we may be able to do better if we can find the MECB without listing all maximal bicliques. Although algorithms do exist for the problem of enumerating all maximal bicliques in general graphs [3] and some special classes of bipartite graphs, specifically, chordal bipartite graphs [20], and bipartite domino-free graphs [4], to the author's best knowledge, no algorithms has been reported for finding the MECB in any special classes of bipartite graphs. Two highly relevant studies that provide methods for enumerating all maximal bicliques will be reviewed below; one of them deals with general graphs, and the other one concerns chordal bipartite graphs, which constitute a super class of the convex bipartite graphs.

2.3.1 The consensus algorithms

According to [29], the consensus algorithm was originally developed by Malgrange to find the maximal submatrices consisting only of one's in a binary matrix – a process that is clearly equivalent to enumeration of all maximal bicliques in a bipartite graph. Alexe et al. [3] extended this method to general graphs and modified it to improve its time bound. The following is a description of Alexe et al.'s algorithms.

Let G be graph, and let $B_1 = (X_1, Y_1)$ and $B_2 = (X_2, Y_2)$ be two bicliques of G . B_1 is said to *absorb* or *contain* B_2 if $X_2 \subseteq X_1$ and $Y_2 \subseteq Y_1$, or if $X_2 \subseteq Y_1$ and $Y_2 \subseteq X_1$. If $X_2 \cap X_1 \neq \emptyset$, then we call $(X_2 \cap X_1, Y_2 \cup Y_1)$ one of the consensus of B_1 and B_2 . It is easy to see that $(X_2 \cap X_1, Y_2 \cup Y_1)$ is a biclique itself. Similarly, $(Y_2 \cap Y_1, X_2 \cup X_1)$, $(X_2 \cap Y_1, Y_2 \cup X_1)$, $(Y_2 \cap$

$X_1, X_2 \cup Y_1$) are possible consensuses of B_1 and B_2 . Note that, in case of bipartite graphs, the latter two pairs of subsets do not exist.

2.3.1.1 The basic consensus algorithm

The *basic consensus algorithm* starts with a collection C of bicliques that covers the edges set of G , repeatedly applies two transformations (see below) on C , and stops when none of the operations is applicable. The two transformations are:

- (i) Absorption: For any two bicliques B_1 and B_2 in C , if B_1 absorbs B_2 , remove B_2 from C ;
- (ii) Consensus adjunction: For any two bicliques B_1 and B_2 in C , if any of the consensus of B_1 and B_2 exists and is not absorbed by a biclique that is already in C , then add it to C .

Let $\hat{C}$ be the collection of bicliques obtained upon the termination of the algorithm, according to Theorem 1 of [3], $\hat{C}$ consists of all maximal bicliques of G . This is easy to see, as no more absorption indicates that all elements in $\hat{C}$ are maximal, and no more consensus adjunction means all maximal bicliques in G have been discovered. Let N be the number of vertices in G and β be the number of maximal bicliques in G , the time complexity of the basic consensus algorithm could be exponential in N and β [3]. Apparently, this algorithm is not viable for any problem of practical size.

2.3.1.2 The modular consensus algorithm

The first modified version of the basic consensus algorithm is called the *modular consensus algorithm (MCA)*. The basic idea is to only add maximal bicliques to the

collection C following a consensus adjunction operation, so that the size of C is at most β at any time. The *MCA* algorithm finds all maximal biclique in G in time $O(N^2\beta^2)$.

ALGORITHM MCA

Input: A graph G .

Output: a list of all maximal bicliques in G .

1. Start with a list C_0 of at most $|E|$ maximal bicliques that cover the edges of the graph G . Let $C = C_0$.
2. Repeat: For any consensus B of B_1 and B_2 , if B is not absorbed by any member of C , extend B to a maximal biclique B' , and add B' to C .

2.3.1.3 The modular input consensus algorithm

A further improvement is achieved in the following *modular input consensus algorithm* (*MICA*), which differs from the *MCA* in two aspects. First, it starts with a set of maximal bicliques that cover the stars, instead of edges, of G . Second, a consensus conjunction operation always involves at least one maximal biclique in the initial collection C_0 . The result of these modifications is an $O(N^3\beta)$ time algorithm. Since β is typically much larger than N , the *MICA* is more efficient than the *MCA*.

ALGORITHM MICA

Input: A graph G .

Output: a list of all maximal bicliques in G .

1. Start with a list C_0 of at most N maximal bicliques that cover the stars of the graph G . Let $C = C_0$.

2. Repeat: For any consensus B of $B_1 \in C_0$ and $B_2 \in C$, if B is not absorbed by any member of C , extend B to a maximal biclique B' , and add B' to C .

The space complexities of MCA and MICA are not discussed in [3]. From the above descriptions, we believe their storage requirements are $O(E)$.

To the author's particular interest, Alexe et al. [3] noted that for convex bipartite graphs, the *MECB* can be found in polynomial time. Since the number of maximal bicliques in a convex bipartite graph, with n vertices on the non-convex side and m vertices on the convex side, is $O(m^2)$, it is not hard to see that, using the *MICA*, the *MECB* in a convex bipartite graph can be found in time $O((n+m)^3 m^2)$.

2.3.2 Enumeration of all maximal bicliques in chordal bipartite graphs

Convex bipartite graphs form a subclass of chordal bipartite graphs. Therefore, some results from chordal bipartite graphs as relevant to the current study are reviewed next.

The following two definitions regarding $(0, 1)$ matrices have been introduced in [20]:

- (i) An ordered $(0, 1)$ -matrix is *Γ -free* if it has no ordered submatrix of the form $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.
- (ii) A *doubly lexical ordering* of a $(0, 1)$ matrix is an ordering of the columns and of the rows such that both the columns and the rows, as vectors, are lexically non-decreasing. (A vector x precedes a vector y if in the last different entry, x has a '0' and y has a '1'.)

A graph is called *chordal bipartite graph* if it is bipartite and each cycle of length at least six has a chord. Kloks et al. [20] proposed a maximal biclique listing algorithm that runs in time $O(N + |E|)$ given a chordal bipartite graph represented as a bipartite adjacency matrix which is Γ -free. When Kloks' algorithm was proposed, the best way for obtaining a Γ -free ordering was to generate a *doubly lexical ordering* of rows and columns of the bipartite adjacency matrix given a chordal bipartite graph; and the algorithm for producing a *doubly lexical ordering* ran in time $O(\min(|E| \log N, N^2))$ [25,33]. Recently, Uehara [36] demonstrated that a Γ -free bipartite adjacency matrix can be computed in $O(N + |E|)$ time for a chordal bipartite graph. Thus, given a chordal bipartite graph G , the current best algorithm can list all maximal bicliques in G in time $O(N + |E|)$. Intuitively, the same upper bound exists for the class of convex bipartite graphs, which is a subclass of chordal bipartite graphs. The maximal biclique listing algorithms and related characterizations for chordal bipartite graphs are briefly summarised below.

2.3.2.1 Perfect edges without vertex elimination ordering (PEWVEO)

The maximal bicliques enumeration algorithm of Kloks and Katsch [20] is based on an ordering called *perfect edges without vertex elimination ordering* (PEWVEO). Let $G = (A, B, E)$ be bipartite graph. The edge $(x, y) \in E$ is called a *bisimplicial edge* if $\lambda(x) \cup \lambda(y)$ induces a biclique in G . Let $e_1, e_2, \dots, e_{|E|}$ be an ordering of E . For $i = 0, 1, \dots, |E|$ define the subgraph $G_i = (A, B, E_i)$ by $E_0 = E$, $E_i = E_{i-1} \setminus \{e_i\}$ for $i \geq 1$. The ordering $e_1, e_2, \dots, e_{|E|}$ is a *perfect edges without vertex elimination ordering* for G if each edge e_i is a bisimplicial edge in G_{i-1} . A bipartite graph G is chordal bipartite if and only if there is a PEWVEO for the set of edges of G (See Lemma 4 of [20]).

Kloks [20] and Uehara [36] provided algorithms for computing the PEWVEO given a chordal bipartite graph that is represented as a Γ -free bipartite adjacency matrix. Their algorithms are strikingly similar, corroborating the notion that what matters is only the Γ -free property of the bipartite adjacency matrix. Assuming the graph is stored as adjacency lists, the following algorithm appeared in [20] apparently runs in time $O(N + |E|)$.

ALGORITHM PEWVEO

Input: a chordal bipartite graph G that is represented using a Γ -free adjacency matrix.

Output: a PEWVEO in G .

FOR row $i = 1$ to n , DO

In increasing order of nonzero entries M_{ij} in row i , DO

put the edge (x_i, y_j) in *pewveo*, and set M_{ij} to 0

END

END

2.3.2.2 Listing the maximal bicliques of a chordal bipartite graph

With the PEWVEO on hand, we can now proceed to the algorithm for listing all maximal biclique in a chordal bipartite graph. From the definition of PEWVEO, we know that each e_i in the ordering $e_1, e_2, \dots, e_{|E|}$ defines a biclique with respect to G_{i-1} , the task now is to find if the biclique associated with e_i is maximal.

Let's change the notation for an edge from e to (x, y) for the sake of convenience. Let M_{ij} be a nonzero entry of the Γ -free bipartite adjacency matrix M . We denote by (X_{ij}, Y_{ij}) the biclique associated with the edge (x_i, y_j) with respect to the ordering $e_1, e_2, \dots, e_{|E|}$,

namely, $X_{ij} = \{x_k \mid k \geq i, M_{kj} = 1\}$, $Y_{ij} = \{y_l \mid l \geq j, M_{il} = 1\}$. Since each entry of M represents an edge, which in turn is associated with a biclique, a maximal biclique must correspond to an edge which itself is an entry in M . We call a nonzero entry M_{ij} maximal if (X_{ij}, Y_{ij}) is a maximal biclique. For every nonzero entry M_{ij} , we define:

$$d_{ij} = \sum_{k=i}^n M_{kj}, \text{ and } r_{ij} = \sum_{l=j}^m M_{il}$$

where $n = |A|$, $m = |B|$ for a bipartite graph $G = (A, B, E)$. Notice that d_{ij} and r_{ij} are not defined if $M_{ij} = 0$. By Lemma 16 of [Kloks, 1995 11 /id], A nonzero entry M_{pq} of the Γ -free bipartite adjacency matrix M is maximal if and only if

$$\forall_{k < p} [r_{kq} < r_{pq}] \text{ and } \forall_{l < q} [d_{pl} < d_{pq}]$$

These conditions guarantee that for each maximal entry M_{pq} , there is no other entry M_{ij} with $i \leq p$ and $j \leq q$ such that $X_{pq} \subseteq X_{ij}$ and $Y_{pq} \subseteq Y_{ij}$. The following algorithm, referred to as CHORDAL in the remainder of this study, for listing all maximal bicliques of a chordal bipartite graph G that is given as a Γ -free bipartite adjacency matrix M , was designed based on Lemma 16 of [20].

ALGORITHM CHORDAL (by Kloks and Kratsch)

Input: a chordal bipartite graph G that is represented using a Γ -free adjacency matrix.

Output: a list of all maximal bicliques in G .

1. Compute r_{ij} and d_{ij} for all nonzero entries of a Γ -free bipartite adjacency matrix M ;
2. For row $i = 1$ to n , do: Scan through all nonzero entries in increasing order of j and label M_{ij} with Δ if $d_{ij} > \max (d_{il} \mid l < j)$;

3. For column $j = 1$ to m , do: Scan through all nonzero entries in increasing order of i and label M_{ij} with P if $r_{ij} > \max (r_{kj} \mid k < i)$;
4. Output (x_i, y_j) for all such entries M_{ij} that are labelled both Δ and P .

Knowing (x_i, y_j) , the corresponding maximal biclique (X_{ij}, Y_{ij}) can be listed easily in the order of PEWVEO. Clearly, Algorithm CHORDAL runs in time $O(N + |E|)$ and uses $O(|E|)$ space.

Chapter 3 Algorithms for finding the MECB in convex bipartite graphs

Given a B -convex bipartite graph $G = (A, B, E)$, by the convexity of the ordering of B , G has a unique structural property: for each vertex a in A , $\lambda(a)$ is an interval on B . We argued that more efficient algorithms for finding MECB could be developed by fully exploiting this structure of convex bipartite graphs. In this chapter, we present two algorithms for finding the MECB in convex bipartite graphs.

3.1 PROBLEM TRANSFORMATIONS

Throughout the remainder of the paper, for a bipartite graph $G = (A, B, E)$, denote $|A| = n$, $|B| = m$.

3.1.1 Graph to interval transformation

The problem of finding MECB $K = (X, Y)$ in a bipartite graph $G = (A, B, E)$ with $A = (a_1, a_2, \dots, a_n)$ and $B = (b_1, b_2, \dots, b_m)$, is equivalent to the following optimization problem: Given a collection $A = \{A_1, \dots, A_n\}$ of finite sets, find a subset $X = \{A_x: 1 \leq x \leq n\}$ where $A_x \in A$ with $\cap A_x = Y$, such that $|X| \times |Y|$ is maximized. The reduction is as follows: G is a

B -convex bipartite graph, and therefore, for $1 \leq i \leq n$, A_i is an interval in the range $B = [1, m]$ [24]. Since Y is the intersection of intervals in X , Y can be viewed as the largest interval on B satisfying $Y \subseteq A_x$ for all $A_x \in X$ [3]. We further reduce the problem of finding MECB in convex bipartite graphs by transforming it into the following geometrical optimization problem: Given a set of 1-dimensional intervals $I_A = \{I_1, \dots, I_n\}$, find a subset X of I_A , such that $|X| \times \text{length}(Y)$ is maximized, where Y is the largest interval satisfying $Y \subseteq I_x$ for all $I_x \in X$ (Figure 4). Clearly, computation of interval enclosure is central to this problem.

Figure 4. An interval representation of bipartite graphs depicted in Figure 1. The labelling corresponds to that of Figure 1. The intervals shown between the dotted vertical lines indicate the bicliques shown in Figure 1.

3.1.2 Interval to point transformation

In Section 2.2.1, we have mentioned that the interval enclosure relationship in a 1-dimensional space is equivalent to the point dominance relationship in a 2-dimensional space [17]. Indeed, given two intervals $I_1 = [i_1, j_1]$ and $I_2 = [i_2, j_2]$ with $I_2 \subseteq I_1$, if we transform them into a pair of 2-dimensional points $p_1 = (-i_1, j_1)$ and $p_2 = (-i_2, j_2)$, clearly, $p_1 \succeq p_2$. In order to avoid negative values, for a pair of points p_1 and p_2 representing two intervals, we redefine the dominance relationship as follows: $p_1 \succeq p_2$, if $x_1 \leq x_2$ and $y_1 \geq$

y_2 . If we define an $m \times m$ grid with the values on the x - and y -axis represents the left endpoints and right endpoints of intervals, respectively; and define a point $p = (x, y)$ as valid point if $y \geq x$, then each grid point above the line $y = x$ will be valid and represent a possible Y , and thus is associated with a possible biclique; conversely, each possible biclique is associated with a unique valid point on the grid. The points corresponding to intervals in I_A are called input points (Figure 5 shows a grid representation of the intervals in Figure 4). All points below the line $y = x$ are invalid and are excluded from further consideration. For an arbitrary valid point p , define dominance number of p , denoted by $DOM(p)$, as the number of input points that dominate p , then $DOM(p)$ equals the number of input points within the northwest quadrant of p (inclusive). Let $S(p)$ or $S(x, y)$ denote the size of the biclique associated with $p = (x, y)$, then $S(p) = S(x, y) = (y - x + 1) \times DOM(p)$. Now, the MECB problem is further reduced to: find a point p on the $m \times m$ grid such that $S(p)$ is maximized.

Figure 5. Grid representation of intervals. The intervals in Figure 4.a are mapped onto a 5×5 grid. The filled circles represent input points. The figure in the parenthesis beside each point denotes the S value of the point.

3.2 ALGORITHM DESCRIPTION

3.2.1 Algorithm BICLIQUE_DOM

An $O(m^2 + n)$ algorithm is readily conceivable following an interval-to-point transformation. Specifically, we can find the MECB by scanning across the grid and computing $S(p)$ at each valid point p . The scanning is performed row-wise (or column-wise) starting at the top-left corner, using an array of size m in order to keep track the numbers of dominance points. In this way, $S(p)$ can be found in time $O(1)$ at each grid point p .

ALGORITHM BICLIQUE_DOM(I_A)

Input: a convex bipartite graph $G = (A, B, E)$. Assume G is represented with an array of intervals I_A .

Output: the maximum edge cardinality biclique $K = (X, Y)$ and the size of K .

1. Transform intervals in I_A into n 2-dimensional points and sort them by decreasing y-coordinates. For each subset of points sharing the same y-coordinate, sort them again by increasing x-coordinates.
2. Define an array DOM of size m and initialize its contents to 0. Define Index = 0.

/ Index points to a position in the input point array */*

At each row j in m to 1:

$SUM_{row} = 0$.

For i in 1 to j :

If grid point (i, j) corresponds to an point in the input array at the position Index ,

then $SUM_{row} \leftarrow SUM_{row} + 1$,

$$DOM[i] \leftarrow DOM[i] + SUM_{row},$$

$$S(i, j) \leftarrow DOM[i] \times (j - i + 1).$$

$$Index \leftarrow Index + 1;$$

$$\text{If } S(i, j) > MAX, \text{ then } MAX \leftarrow S(i, j), Y \leftarrow [i, j].$$

3. Perform a linear scan of the list I_A to identify X . $K = |X| \times |Y|$.

END *BICLIQUE_DOM*

Lemma 3.2.1.1: *Algorithm BICLIQUE_DOM finds the maximum biclique in a convex bipartite graph.*

Proof: Each possible biclique is associated with a valid point on the $m \times m$ grid.

Algorithm *BICLIQUE_DOM* examines all valid points on the grid. ■

Lemma 3.2.1.2: *The time complexity of Algorithm BICLIQUE_DOM is $O(m^2 + n)$, and the space complexity is $O(m + n)$.*

Proof: In step 1, interval to point transformation takes $O(n)$ time. Sorting these input points by x - and y -coordinates can be done in $O(n)$ time using Radix-sort given that these coordinates are integers and their range is known. As stated above, all points above the line $y = x$ are visited in step 2; thus, the total number of points visited is $O(m^2)$. At each point p , only constant time is required to compute the $S(p)$. The step 3 requires $O(n)$ time and is performed only once. Therefore, the total time complexity is $O(m^2) + O(n) = O(m^2 + n)$. The space complexity is $O(m + n)$ due to the sorted input array and the array *DOM*. Note that the grid is implicit; only a row of the grid is maintained in the memory at any time. ■

Theorem 3.2.1.3: *Algorithm BICLIQUE_DOM finds the MECB in a convex bipartite graph in $O(m^2 + n)$ time and in $O(m + n)$ space.*

Proof: These results follow Lemma 3.2.1.1 and Lemma 3.2.1.2. ■

3.2.2 Algorithm BICLIQUE_DOMA

When $|A| \ll |B|$, it may not be necessary to visit all points above the line $y = x$. Thus the question arises if the $O(|B|^2 + |A|)$ time bound can be improved in this case?

Lemma 3.2.2.1: *The number of maximal bicliques in a B-convex bipartite graph is $O(n^2)$.*

Proof: In a B-convex bipartite graph, the Y -side of any maximal biclique must start with a starting vertex, denoted $start(i)$ of $\lambda(a_i)$, and end with a finishing vertex, denoted $finish(j)$ of $\lambda(a_j)$, where $finish(j) \geq start(i)$ for $1 \leq i \leq n$ and $1 \leq j \leq n$. This is easy to explain: assume $K = (X, Y)$ is a maximal biclique and the Y -side of K starts with a vertex v that is not a starting vertex of any $\lambda(a_i)$, then there will be another biclique $K' = (X, Y')$ such that $|Y'| > |Y|$. This contradicts the fact that K is a maximal biclique. Therefore, v must be a starting vertex of some $\lambda(a_i)$. The same argument holds for finishing vertex. There are at most $O(n^2)$ ordered pairs of $(start(i), finish(j))$ on B . Hence, the number of maximal bicliques in a convex bipartite graph is $O(n^2)$. ■

By Lemma 3.2.2.1, an $O(|A|^2)$ solution is possible if we limit the search space to maximal bicliques only. Let G be a convex bipartite graph, we say that a point p on the grid derived from G is maximal if and only if the following two conditions are satisfied: i)

$DOM(p) > 0$; ii) There exists no other point $q \neq p$ on the grid such that $q \succeq p$ and $DOM(q) = DOM(p)$. (Note that a maximal interval Y can be defined analogously.)

Lemma 3.2.2.2: *There is a one-to-one correspondence between a maximal biclique in a convex bipartite graph $G = (A, B, E)$ and a maximal point on the grid derived from G .*

Proof: There are two cases to address in order to show the one-to-one relationship between maximal bicliques in G and maximal points in the corresponding grid.

First case: if p is a maximal point on the grid, then p corresponds to a maximal biclique in G . Let p be a maximal point and let $K_p = (X_p, Y_p)$ be the biclique in G associated with p , such that $Y_p \in B$ corresponds to the interval represented by p and $X_p \in A$ consists of all vertices whose neighbourhoods enclose Y_p (note that $|X_p| = DOM(p)$). Assume K_p is not a maximal biclique, then there must exist a biclique $K_q = (X_q, Y_q) \neq K_p$ in G such that $Y_q \supseteq Y_p$ and $X_q \supseteq X_p$. Let q be the grid point associated with K_q . The first relationship, $Y_q \supseteq Y_p$, implies that $q \succeq p$; and $X_q \supseteq X_p$ implies $|X_q| \geq |X_p|$, which is equivalent to $DOM(q) \geq DOM(p)$. This contradicts the condition that p is a maximal point. Thus, K_p is a maximal biclique because there is no biclique that contains K_p in G .

Second case: if $K_p = (X_p, Y_p)$ is a maximal biclique in G then the corresponding point p in G is a maximal point. Let $K_p = (X_p, Y_p)$ be a maximal biclique in G and let p be a grid point associated with K_p . Then, since K_p is a maximal biclique, it implies that $|X_p| > 0$, and therefore $DOM(p) > 0$. If p is the top-left corner point $(1, |B|)$ of the grid, p qualifies for a maximal point instantly as there is no other point dominates p . Otherwise, suppose that p is not maximal, then there must exist another point $q \neq p$ on the grid such that $q \succeq p$

and $DOM(q) = DOM(p)$. Let D_q and D_p be the sets of input points that dominate q and p , respectively. It is easy to verify that $D_q = D_p$ when $q \succeq p$ and $DOM(q) = DOM(p)$. ($q \succeq p$ implies that $D_q \subseteq D_p$, and $DOM(q) = DOM(p)$ is equivalent to $|D_q| = |D_p|$, therefore, $D_q = D_p$.) Thus, existence of q implies that there is a biclique $K_q = (X_q, Y_q) \neq K_p$ in G such that $Y_q \supset Y_p$ and $X_q = X_p$. This contradicts the condition that K_p is a maximal biclique. Therefore, p must be a maximal point. ■

Observation 3.2.2.3: *Let $p = (x_p, y_p)$ be a point on the grid. Then, p is maximal if and only if there exist input points $l = (x_l, y_l)$ and $t = (x_t, y_t)$ such that $y_l = y_p$, $x_l \leq x_p$ and $y_t \geq y_p$, $x_t = x_p$.*

Proof: By the definition of maximal point, a grid point that corresponds to an input point satisfies the above conditions and qualifies for maximal point automatically. (In this case, the points l and t are the same as p , i.e., $y_l = y_p$, $x_l = x_p$ and $y_t = y_p$, $x_t = x_p$.) So we only have to show that this observation is also true for grid points that do not correspond to any input points. First, we prove that if there are input points l and t satisfying the above conditions, then, the grid point p is maximal. First, it is easy to see that $DOM(p) > 0$ due to the existence of l and t . By the definition of the point dominance relationship, any grid point q that dominate p and $q \neq p$ will lie above and/or to the left of p on the grid, thus we will have $DOM(q) < DOM(p)$, as $DOM(q) \leq DOM(p) - 1$. Therefore, p is maximal.

Next, we show that if p is maximal and is not associated with any input points, then there exist l and t satisfying the above conditions. Assume p is maximal and there is no input point $l = (x_l, y_l)$ satisfying $y_l = y_p$, $x_l < x_p$, then there exists a grid point q immediately

above p , namely $q = (x_p, y_p + 1)$, satisfying $q \succeq p$, $q \neq p$ and $DOM(q) = DOM(p)$. This contradicts the condition that p is maximal. Therefore, there must exist a point $l = (x_l, y_l)$ satisfying $y_l = y_p$, $x_l < x_p$. Based on similar arguments, there must exist a point $t = (x_t, y_t)$ satisfying $y_t > y_p$, $x_t = x_p$. ■

Based on Observation 3.2.2.3, we modify the grid so that each row (and each column) contains at least one input point. This can be done as follows: sort the input points by distinct x -coordinates in ascending order (and do the same with y -coordinates), and replace the original coordinates with their ranks in their respective sorted sequences; then map them onto a $n \times n$ grid defined by the ranks. Now, using the approach of Algorithm *BICLIQUE_DOM*, we can find the MECB in $O(n^2)$ time and $O(n)$ space. However, it is still possible that a number of valid points on the grid are not associated with maximal bicliques. In the remainder of this section, we will show an algorithm that examines maximal bicliques only using a plane-sweep approach as outlined below.

The grid is swept from left to right. Let x_o, y_o be the original values of x, y of a transformed point (x, y) , and a point is valid if its $y_o - x_o \geq 0$. When the sweep-line pauses at each column, we first insert all input points in the current column into a list, called *y-list*, sorted by the y -coordinates in descending order, then we scan through *y-list* starting with the first input point in the column, until we reach the point with $x_o = y_o$. Let $w = (x_w, y_w)$ be a point in column j , and let $p = (x_p, y_p)$ be a point in *y-list* such that $y_p = y_w$ and x_p is the largest value that is smaller than or equal to x_w , then the rank of p in *y-list* gives $DOM(w)$. This is easily seen: as we are sweeping from left to right (in increasing x -

coordinates) and inserting points top-down (in decreasing y -coordinates) at each column j , all points q in front of p (including p) in y -list have the following property: $x_q \leq x_w$ and $y_q \geq y_w$, i.e., $q \succeq w$. Conversely, all input points with the above property must have been inserted in front of p . Given $DOM(w)$, the value $S(w) = (y_o - x_o + 1) \times DOM(w)$ can be computed in constant time.

ALGORITHM BICLIQUE_DOMA (I_A)

Input: a convex bipartite graph $G = (A, B, E)$. Assume G is represented with a list of intervals I_A .

Output: the maximum edge cardinality biclique $K = (X, Y)$ and the size of K .

1. Sort intervals in I_A by ascending left endpoints and right endpoints. Transform these intervals into 2-dimensional points with ranks of left endpoints and right endpoints as x - and y -coordinates.

2. At each column j in 1 to n :

2.1 Starting from the topmost input point, insert all input points in column j into a y -list, which is sorted in descending y -coordinates.

2.2 Starting from the topmost input point in j , for each y of highest rank in y -list, do the following until $y_o = x_o$:

$$S(x, y) \leftarrow rank(y) \times (y_o - x_o + 1).$$

If $S(x, y) > MAX$, then

$$MAX \leftarrow S(x, y), Y \leftarrow [x_o, y_o].$$

3. Perform a linear scan of the list I_A to identify X . $K = |X| \times |Y|$.

END BICLIQUE_DOMA

Lemma 3.2.2.4: *Algorithm BICLIQUE_DOMA computes the size of every maximal biclique in a convex bipartite graph.*

Proof: At each column of the grid, the algorithm starts at the topmost input point in the column, and a grid point below the topmost input point will be examined (i.e., $S(x, y)$ is computed) if and only if it is a maximal point. This is achieved through the guidance of the y -list: if a grid point $w = (x_w, y_w)$ is not maximal, then the value y_w will not be in the y -list, thus w will not be examined. Therefore, only maximal bicliques were examined. Conversely, if a point w is maximal, it must lie below the topmost input point in j , and there must be a point p in y -list such that $x_p \leq x_w$ and $y_p = y_w$. Thus w will be examined. Therefore, Algorithm BICLIQUE_DOMA visited all maximal points, and thus examined all maximal bicliques by Lemma 3.2.2.2. ■

Lemma 3.2.2.5: *Algorithm BICLIQUE_DOMA takes, in the worst case, $O(d_{\max}(B) n)$ time and $O(n)$ storage space.*

Proof: In step 1, sorting the input intervals by left endpoints and right endpoints can be done in linear time for reasons stated in proof of Lemma 3.2.1.2. Transformation of the intervals takes $O(n)$ time. If we let $d_{\max}(B)$ be the maximum degree of B , we will have $d_{\max}(B) = \max(DOM(p) \mid (x_p)_o = (y_p)_o, 1 \leq x_p \leq n)$, where $(x_p)_o$ and $(y_p)_o$ are the original values of x_p and y_p , respectively. Clearly, for an arbitrary valid point p on the grid, $DOM(p) \leq d_{\max}(B)$. Therefore, inserting an element into y -list takes time $O(d_{\max}(B))$ when the order of insertion detailed in the step 2.1 is followed. At each column j , $1 \leq j \leq n$, at most $d_{\max}(B)$ query points are examined. Note that each query point can be examined in

constant time. Once the interval Y is located, the identities of the input intervals that enclose Y can be determined trivially in $O(n)$ time. Therefore the total complexity of the Algorithm BICLIQUE_DOMA is $O(d_{\max}(B)n + n) = O(d_{\max}(B)n)$. Besides the input list I_A , only an auxiliary list of size n is used, therefore, the total storage space is $O(n)$. ■

Theorem 3.2.2.6: *Algorithm BICLIQUE_DOMA finds the MECB in a convex bipartite graph in $O(d_{\max}(B)n)$ time and in $O(n)$ space.*

Proof: These results follow Lemma 3.2.2.4 and Lemma 3.2.2.5. ■

Chapter 4 Algorithms for finding the MECB in doubly convex bipartite graphs and bipartite permutation graphs

The class of doubly convex bipartite graphs is a subclass of convex bipartite graphs and a super class of the bipartite permutation graphs, i.e.,

$$\text{bipartite permutation graph} \subseteq \text{doubly convex bipartite graph} \subseteq \text{convex bipartite graph}.$$

For a doubly convex bipartite graph $G = (A, B, E)$, a much stronger constraint on vertex ordering exists, namely, G is convex on both A and B . This makes it possible to represent a doubly convex bipartite graph as a two dimensional geometric object within which an equivalent of the MECB problem can be defined. In this chapter, we show that the MECB problem in doubly convex bipartite graphs and bipartite permutation graphs can be reduced to a geometrical problem, namely the *maximum area axis-parallel rectangle* (MAAPR) problem. The geometrical problem can then be solved efficiently through application of advanced matrix searching techniques.

4.1 Finding the MECB in doubly convex bipartite graph

A polygon P is an *orthogonally convex rectilinear polygon* if P is convex on a pair of orthogonal coordinate axes, which are horizontal and vertical without loss of generality,

and the edges of P are all aligned with these axes. A doubly convex bipartite graph can be represented as an orthogonally convex rectilinear polygon if its interval representation is extended into a two-dimensional space [24,39]. This section shows that the MECB problem in the doubly convex bipartite graph can be reduced to the problem of finding the MAAPR in an orthogonally convex rectilinear polygon. This problem can be solved in $O(\alpha(n)n)$ time, where $\alpha(n)$ is the inverse of Ackermann's function [19]. It is a very slowly growing function, and for practical applications, $\alpha(n) \leq 5$ ($\alpha(n) \leq 5$ when $n \leq 2^{65,536}$).

Figure 6. Graph to interval transformation. A doubly convex bipartite graph on the left is represented as an ordered list of intervals on the right. The 1-dimensional intervals are mapped onto a 2-dimensional space with π^A and π^B as the y- and x-coordinates, respectively.

4.1.1 Transforming a doubly convex bipartite graph into an orthogonally convex rectilinear polygon

If a doubly convex bipartite graph is represented as a bipartite adjacency matrix, all entries that contain 1's will naturally form an orthogonally convex rectilinear polygon

whose vertex chain can be defined easily with row and column indices of the matrix. Apparently, a bipartite adjacency matrix will take $O(nm)$ space. However, if the doubly convex bipartite graph is given as an ordered list of n intervals, only $O(n)$ space and time is required to perform the transformation, as described below.

Figure 7. Interval to polygon transformation. **a):** The 2-dimensional intervals shown in Figure 6.b are expanded into rectangles of height 1. **b):** An orthogonally convex rectilinear polygon formed by merging the rectangles on the left.

Lemma 4.1.1.1: An ordered list of n intervals representing a doubly convex bipartite graph $G = (A, B, E)$, where $n = \min(|A|, |B|)$, can be transformed into an orthogonally convex rectilinear polygon in $O(n)$ space and time.

Proof: Given a doubly convex bipartite graph $G = (A, B, E)$ as an ordered list of intervals, we can first transform the set of ordered intervals into two-dimensional horizontal line segments using the ordering information of π^A and π^B as y -axis and x -axis coordinates, respectively. For $1 \leq k \leq n$ and $1 \leq s < t \leq m$, the k^{th} interval $[s, t]$ will be changed into the line segment $[(s, k), (t, k)]$ (Figure 6.b). Then expand each line segment downwards 1 unit length into a rectangle of unit height. A line segment $[(s, k), (t, k)]$ will

be turned into a rectangle with top-left corner at (s, k) and bottom-right corner at $(t, k-1)$ (Figure 7.a). Then merge all stacking rectangles to form an orthogonally convex rectilinear polygon (recall that both π^A and π^B are convex), whose vertex chain consists of two sub-chains, i.e., *left sub-chain* and *right sub-chain*, composed of the top-left and bottom-right corners of the original rectangles, respectively (Figure 7.b). Each sub-chain contains n vertices, and thus P consists of $2n$ vertices in all. Apparently, the transformation process can be done in time $O(n)$, as each interval can be turned into a rectangle in constant time and each interval is only processed once. The space complexity is $O(n)$ also, since the resulting polygon chain has only $2n$ vertices. ■

For example, the interval representing $\lambda(a_4) = [b_1, \dots, b_6]$ in Figure 6.a will be firstly changed into $[(1,4), (6,4)]$ (Figure 6.b). Then the line segment $[(1,4), (6,4)]$ will be transformed into a rectangle with top left corner at $(1,4)$ and bottom right corner at $(6,3)$ (Figure 7.a). Finally, the rectangle will be merged with other rectangles to form an orthogonally convex rectilinear polygon shown in Figure 7.b.

Lemma 4.1.1.2: *The problem of finding the MECB in a doubly-convex bipartite graph is equivalent to the problem of finding the MAAPR in the orthogonally convex rectilinear polygon produced by the above construction procedure.*

Proof: By Lemma 4.1.1.1, if a doubly convex bipartite graph $G = (A, B, E)$ with $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_m\}$, is represented using a set of ordered intervals, then following the above transformation steps, the rectilinear polygon P constructed from G will be an orthogonally convex rectilinear polygon. Let $R = ((i, j); (k, l))$ be an axis-parallel

rectangle contained in P , with top-left and bottom-right corners of R at (i, l) and (j, k) , respectively. Then R will correspond to a biclique $K = (X, Y)$ in the original doubly convex bipartite graph G , with $X = \{b_i, \dots, b_j\}$ and $Y = \{a_k, \dots, a_l\}$. Conversely, any biclique $K = (X, Y)$ with edge cardinality $|X| \times |Y|$ in G will be mapped to an axis-parallel rectangle R in P with width $W(R) = |X|$ and height $H(R) = |Y|$. An intuitive way of appreciating the correspondence between a biclique in G and a rectangle in P is to consider P as the area occupied by non-zero entries of a bipartite adjacency matrix M of G , with each 1×1 sub-area in P corresponding to a non-zero entry in M which, in turn, represents an edge between a vertex in A and a vertex in B of G .

Now, let us first show that MECB maps to MAAPR. Let $K = (X, Y)$ be a MECB in G and it is transformed to an axis-parallel rectangle R in P . Suppose R is not the MAAPR in P , then there must exist another axis-parallel rectangle R' in P such that $W(R') \times H(R') > W(R) \times H(R)$. Since each axis-parallel rectangle in P corresponds to a biclique in G , this implies that there exists a biclique $K' = (X', Y')$ in G such that $|X'| \times |Y'| > |X| \times |Y|$. This contradicts the condition that $K = (X, Y)$ is a MECB in G , thus, R is the MAAPR in P . Then, we show that MAAPR maps to MECB. Let an axis-parallel rectangle R be a MAAPR in P , and it represents a biclique $K = (X, Y)$ in G . If $K = (X, Y)$ is not the MECB in G , then there must be another biclique $K' = (X', Y')$ in G such that $|X'| \times |Y'| > |X| \times |Y|$. Let R' be the image of K' in P , then $W(R') \times H(R') > W(R) \times H(R)$, which contradicts that R is an MAAPR in P , thus $K = (X, Y)$ must be the MECB in G . ■

Theorem 4.1.1.3: *The problem of finding the MECB in a doubly convex bipartite graph G can be reduced to the problem of finding the MAAPR in the orthogonally convex rectilinear polygon P derived from G in linear time.*

Proof: This result follows Lemma 4.1.1.1 and Lemma 4.1.1.2. ■

Figure 8. *This example illustrates the defining points of an orthogonally convex rectilinear polygon, as well as type I and type II maximal rectangles. TLC: top left corner, TRC: top right corner, BLC: bottom left corner, BRC: bottom right corner, LE: left extreme, RE: right extreme.*

4.1.2 Finding the MAAPR in an orthogonally convex rectilinear polygon

According to the Theorem 5.8 of Daniels et al. [9], the MAAPR in an orthogonally convex rectilinear polygon P with n vertices can be found in time $O(\alpha(n)n)$ by using fast matrix searching technique, where $\alpha(n)$ is a very slowly growing inverse of Ackermann's function.

The solution by Daniels et al. [9] is for orthogonally convex polygons, which is more general. In our case, the polygon P evolved from the doubly convex bipartite graph is an orthogonally convex rectilinear polygon and is well shaped; specifically, for each vertex v

in the left vertex chain, there is a vertex u with the same y -coordinate in the right vertex chain, therefore, there is no need for adding special vertices by projecting either vertical or horizontal rays. It has been suggested in [9,10] that the same time complexity holds for orthogonally convex rectilinear polygon (also called orthogonally convex orthogonal polygon). However, no detailed solution was provided for this class of polygon; we now give a detailed account for the problem of finding MAAPR in our case.

4.1.2.1 Two types of maximal rectangle exist in the orthogonally convex rectilinear polygon

Let RE and LE be the right extreme and left extreme vertices of P , respectively (in case where there are more than two extreme vertices on each side of P , define RE as the one with the smallest y -coordinate, and LE is the one with the largest y -coordinate). The left vertex chain can be partitioned into two sub-chains around LE : the top-left sub-chain and the bottom-left sub-chain. In the bottom-left sub-chain, all vertices are below the leftmost point of the polygon, and as the x -coordinates of the vertices increase, the y -coordinates decrease. Whereas in the top-left sub-chain, all vertices are located above the leftmost point of the polygon, and both x -coordinates and y -coordinates increase monotonically. The right vertex chain can be divided into top-right sub-chain and bottom-right sub-chain in a similar way with the RE , as the separating point.

Let us call an axis-parallel rectangle R contained in an orthogonally convex rectilinear polygon P *maximal rectangle* if every sides of R touch the boundary of P . It is easy to see that the MAAPR must be one of the maximal rectangles in P . As P is orthogonally convex and rectilinear, as shown in Figure 8 there are only two types of maximal

rectangles in terms of the number of contacts with P : type-I touches three or four sub-chains and type-II touches only two opposing sub-chains. As we are from now on only concerned with the maximal rectangles, we can ignore all vertices that are on inward corners of the polygon and only consider those at the outward corners of the P .

All type-I maximal rectangles can be found in linear time as follows (see Appendix C in [9]). For any vertex v , let $v.x$ and $v.y$ denote the x and y coordinates of v , respectively. For all pairs of vertices (v_l, u_l) below LE such that v_l belongs to bottom-left sub-chain and u_l belongs to bottom-right sub-chain and $v_l.y = u_l.y$, project rays upwards. Suppose the ray from v_l and u_l exit the polygon at points v_2 and u_2 , respectively, if $u_2.y < v_2.y$, then (v_l, u_2) defines a maximal rectangle; otherwise, (v_2, u_l) does so. The maximal rectangles involve vertices above RE can be found similarly, except that the rays have to be projected downwards now.

4.1.2.2 The implicit area matrix M is totally monotone

However, the type-II maximal rectangles cannot be examined in such a straightforward way as described above for the type-I maximal rectangles. Without loss of generality, let us only consider the top-left sub-chain denoted by *TL-chain* and the bottom-right sub-chain denoted by *BR-chain* for now.

Let us number the vertices in the *TL-chain* by increasing y -coordinates and do the same for the vertices in the *BR-chain*. For each pair of vertices v_i, u_j where $v_i \in BR-chain$ ($1 \leq i \leq n$) and $u_j \in TL-chain$ ($1 \leq j \leq n$), an area function $\text{Area}(v_i, u_j)$ is defined to compute the area of the rectangle with these two vertices as diagonally opposing corners. Since the x -

and y-coordinates of these two vertices are known, each time the area function is called, the answer can be returned in constant time. Note that all type-II maximal rectangles are defined between two opposing vertices, the vertex-edge type maximal rectangles discussed in [9] do not exist for rectilinear polygon.

Suppose we have an *implicit* area matrix M whose entry M_{ij} contains the area of the axis-parallel rectangle with one vertex v_i from the bottom-right sub-chain as its bottom-right corner and another vertex u_j from top-left sub-chain as its top-left corner. Finding the MAAPR amounts to searching the area matrix M for row maxima. The word ‘implicit’ should be emphasized here. This means that the area matrix does not physically exist; instead, entries of the matrix are calculated on the fly on demand, and we do not have to compute all entries for the entire matrix and store them in memory. Otherwise the algorithm will have a time complexity of $O(n^2)$, simply because the area matrix has $O(n^2)$ entries.

Figure 9. An example illustrating the totally monotonicity of the Area matrix.

As these two sub-chains are increasing xy -monotone vertex chains, the area matrix M is totally monotone [9]. Recall that a matrix $M = (M_{ij})$ is totally monotone if for every i, j, k, l such that $i < k, j < l$, $M_{ij} \leq M_{il}$ implies $M_{kj} \leq M_{kl}$ [1]. The total monotonicity of M is illustrated in Figure 13. When all four entries of any 2×2 minor of matrix M are valid entries, as shown in Figure 13, $M_{ij} = \text{Area}(v_i, u_j) = R_1 + R_2 + R_3 + R_4$, $M_{il} = \text{Area}(v_i, u_l) = R_2 + R_4 + R_6$, $M_{kj} = \text{Area}(v_k, u_j) = R_3 + R_4 + R_5$, $M_{kl} = \text{Area}(v_k, u_l) = R_4 + R_5 + R_6 + R_7$, where $R_1, \dots, R_7 \geq 0$. Given $\text{Area}(v_i, u_j) \leq \text{Area}(v_i, u_l)$, we will have $R_1 + R_3 \leq R_6$. From the above, it is obvious that $R_3 \leq R_6 \leq R_6 + R_7$. Altogether, we have: $\text{Area}(v_k, u_j) \leq \text{Area}(v_k, u_l)$.

4.1.2.3 The implicit area matrix M is a totally monotone partial matrix

Two points v and u are rectangularly visible inside a polygon P if the axis-parallel rectangle with diagonal vu is completely inside P [27]. Some pairs of opposing vertices in the TL -chain and BR -chain are not rectangularly visible due to the existence of the bottom-left sub-chain and the top-right sub-chain. Consequently, some entries of M are illegal because the rectangles defined by these vertices is not completely inside P . Conventionally [1,9], those illegal entries with positive values are blanked out, while illegal entries with negative values are kept in the matrix M , as these entries do not violate the total monotonicity property and the row maxima can never be a negative value. Moreover, the time complexity of the searching algorithm only depends on the dimension of the matrix rather than the number of entries. The resulting partial matrix is a totally monotone rising double staircase matrix, whose boundary sequences $\{s_i\}$ and $\{f_i\}$

(refer to Section 2.2.3 for the definition of boundary sequence) are decreasing sequences [9,10]. Note that these boundary sequences can be found in linear time. The row maxima in the rising double staircase matrices cannot be found directly. Daniels et al. (see Appendix D in [9]) have shown that the row maxima in $p \times q$ such matrices can be found in time $O(p\alpha(q) + q)$ through partitioning the double staircase matrix into a set of single staircase matrices in time $O(p + q)$. We have devised an alternative partition scheme that also takes $O(p + q)$ time, as illustrated in Figure 10, and it is used in the implementation of the current study. Given $p \leq n$ and $q \leq n$, this means that the MAAPR in an orthogonally convex rectilinear polygon can be found in time $O(n\alpha(n))$, as all type-II maximal rectangles associated with the bottom-left sub-chain and the top-right sub-chain can be examined analogously in the same order of time and all type-I maximal rectangles can be examined in time $O(n)$.

Figure 10. A schematic drawing of a totally monotone rising double staircase matrix. Two zigzag lines represents the start and finish boundaries of the valid entries. Dashed lines shows how to partition the matrix into rising and reverse rising single staircase matrices.

In Chapter 2, Section 2.2.3, a theoretically slower but easy to implement algorithm by Aggarwal et al. [1] for finding the row maxima in single staircase matrices were described. According to Daniels [9], the $O(p\alpha(q) + q)$ time algorithm by Klawe [19] has large constants.

4.1.3 Algorithm description

ALGORITHM BICLIQUE_MAAPR

Input: A doubly convex bipartite graph $G = (A, B, E)$, represented with an ordered set of intervals. Presumption: Both A and B are ordered such that G is a doubly convex bipartite graph. Without loss of generality, assume $|A| \leq |B|$.

Output: The maximum biclique $K = (X, Y)$ in G and size of K .

1. Transform each interval into a rectangle whose width is 1 and whose length equals the length of the interval.
2. Collect left corners and right corners of all rectangles to form the left vertex chain and right vertex chain, respectively. Divide each vertex chain into two sub-chains around LE and RE .
3. Examine type-I maximal rectangles by sweeping the polygon chains bottom-up and then top-down to find $MAX-I$, the maximum of type-I maximal rectangles.
4. Examine type-II maximal rectangles associated with each pair of opposing sub-chains using a fast matrix searching algorithm introduced in Section 2.2.3 (Also see [1,19] for details) to find $MAX-II$, the maximum of type-II maximal rectangles.
5. $MAAPR \leftarrow \max(MAX-I, MAX-II)$.

6. Find the maximum biclique $K = (X, Y)$ using the x - and y -coordinates of the $MAAPR$. Size of K = area of the $MAAPR$.

END BICLIQUE_ MAAPR

Theorem 4.1.3: *Algorithm BICLIQUE_ MAAPR finds the MECB in a doubly convex bipartite graph in $O(n\alpha(n))$ time and in $O(n)$ space.*

Proof: These results follow Theorem 4.1.1.3 and Theorem 5.8 of [9].■

4.2 Finding the MECB in bipartite permutation graphs

Recall that a bipartite permutation graph $G = (V, E)$ is a permutation graph whose vertex set V can be partitioned into two independent sets A and B . Besides the convex property, the orderings of the vertex sets A and B of a bipartite permutation graph have the additional *enclosure property*: for every pair of vertices $x, y \in A$ (or B), if $\lambda(x) \supseteq \lambda(y)$, then $\lambda(x) - \lambda(y)$ is an interval in the ordering of B (or A) [34].

Lai and Wei [22] introduced the concept of *forward property*: Given a bipartite graph $G = (A, B, E)$, for every pair of vertices a_i, a_j in A such that $a_i \leq a_j$ in π^A , the starting vertex of $\lambda(a_i) \leq$ the starting vertex of $\lambda(a_j)$ and the finishing vertex of $\lambda(a_i) \leq$ the finishing vertex of $\lambda(a_j)$ in π^B . A bipartite graph $G = (A, B, E)$ is *forward convex* if there exists an ordering of $A \cup B$ such that B is convex and the ordering has the forward property. They also showed that bipartite permutation graphs are forward convex (*Theorem 1 of [22]*).

An $O(|E|)$ time recognition algorithm that can be used to produce a forward-convex ordering has been proposed in [22]. It is also noted that a forward-convex labelling (or ordering) definitely has convex and enclosure properties, but the reverse is not true in general. In fact, the reverse *is* true, i.e., if both ordering of A and ordering of B have convex and enclosure properties, then the ordering of $A \cup B$ has the forward-convex property. To put it more precisely, if the ordering of either A or B has convex and enclosure properties, then it is not necessarily true that the ordering of $A \cup B$ has the forward-convex property.

4.2.1 A linear optimal time solution

The point here is that if $G = (A, B, E)$ is a bipartite permutation graph with a forward-convex ordering of $A \cup B$, then its interval presentation can be easily transformed into a *xy-monotone rectilinear polygon*, i.e., an orthogonally convex rectilinear polygon P with only two *xy-monotone* vertex chains, using the procedure described in Section 4.1.1. Next, the MECB in G can be found through finding the MAAPR in P .

Lemma 4.2.1.1: *An ordered list of n intervals representing a bipartite permutation graph $G = (A, B, E)$, where $n = \min(|A|, |B|)$, can be transformed into a *xy-monotone rectilinear polygon* in $O(n)$ space and time.*

Proof: Given the underlying bipartite permutation graph has the forward convex property, the sequence consisting of left endpoints and the sequence consisting of right endpoints of these intervals are *xy-monotone*. Therefore, the vertex chains obtained using the procedure described in Section 4.1.1 must be *xy-monotone* too. ■

Lemma 4.2.1.2: *The problem of finding the MECB in a bipartite permutation graph is equivalent to the problem of finding the MAAPR in the xy -monotone rectilinear polygon produced by the construction procedure described in Section 4.1.1.*

Proof: See the proof of Lemma 4.1.1.2. ■

Theorem 4.2.1.3: *In linear time, the problem of finding the MECB in a bipartite permutation graph G can be reduced to the problem of finding the MAAPR in the xy -monotone rectilinear polygon P derived from G .*

Proof: By Lemma 4.2.1.1 and Lemma 4.2.1.2. ■

The problem of finding the MAAPR in xy -monotone rectilinear polygon is much simpler than that in orthogonally convex rectilinear polygon. There is no type I maximal rectangles involved in this case. As there are only two xy -monotone vertex chains, the area matrix M associated with P is totally monotone (without illegal entries) and is an $n \times n$ square matrix. Recall that the row maxima in a such matrix can be found in time $O(n)$ using the SMAWK algorithm described in Section 2.2.2.

4.2.2 Algorithm description

ALGORITHM BICLIQUE_PERM

Input: A bipartite permutation graph $G = (A, B, E)$, represented with an ordered set of intervals. Presumption: Both A and B are ordered such that G is forward convex. Without loss of generality, assume $|A| \leq |B|$.

Output: The maximum biclique $K = (X, Y)$ in G and size of K .

1. Transform each interval into a rectangle whose width is 1 and whose length equals the length of the interval.
2. Collect left corners and right corners of all rectangles to form the left vertex chain and right vertex chain, respectively.
3. Examine all type-II maximal rectangles associated with a pair of opposing sub-chains using the SMAWK algorithm to find the *MAAPR*.
4. Find the maximum biclique $K = (X, Y)$ using the x - and y -coordinates of the *MAAPR*. Size of K = area of the *MAAPR*.

END BICLIQUE_PERM

Theorem 4.2.1: *The MECB in a bipartite permutation graph $G = (A, B, E)$ can be found in time $\Theta(n)$, where $n = \min(|A|, |B|)$.*

Proof: Given the results of Theorem 4.2.1.3, and Theorem 5.3 of [9], this theorem follows straightforwardly. ■

Chapter 5 Implementation and Experimental Results

To corroborate the theoretical results and to evaluate the efficiency of our algorithms against Algorithm CHORDAL by Kloks et al. [20], several algorithms were implemented and tested in this study. This chapter depicts the relevant details of the implementation and provides a summary of the experimental results.

5.1 Implementation

The following algorithms were implemented in this study:

- (i) Algorithm CHORDAL [20] presented in Section 2.3.2.2.
- (ii) Algorithm BICLIQUE_DOM presented in Section 3.2.1.
- (iii) Algorithm BICLIQUE_DOMA presented in Section 3.2.2.
- (iv) Algorithm BICLIQUE_MAAPR presented in Section 4.2.2.

Algorithm CHORDAL by Kloks and Kratsch [20] was implemented to serve as a reference point for the Algorithm, BICLIQUE_DOM, BICLIQUE_DOMA and BICLIQUE_MAAPR. The Algorithm BICLIQUE_PERM was not implemented to avoid

redundancy. The core procedure of this algorithm is Algorithm SMAWK [2], which was implemented as a subroutine of Algorithm PartialMaxCompute [1], which was in turn implemented as a component of Algorithm BICLIQUE_MAAPR. The consensus algorithms by Alexe et al. [3] were not implemented because they were relatively less relevant than the CHORDAL algorithm. See Table 2 in Section 1.4 for time and space complexities of the implemented algorithms.

5.1.1 System environment

All implementations were performed using Microsoft Windows with C++ as programming language. The C++ compiler embedded in the integrated development environment Microsoft Visual Studio 6.0[®] was employed for compiling and debugging. The machine, which is equipped with a Pentium[®] IV 2.6 GHz CPU and 1 Gigabyte of RAM, runs the Microsoft Windows XP Professional Edition as an operating system.

5.1.2 General design issues

Four data types were defined: Interval, Vertex, MyArray, and Matrix. The Interval class contains the left and right endpoints of a one-dimensional interval. The Vertex class contains the x- and y-coordinates of a point in a two-dimensional space. The MyArray class is a template class containing a dynamic array of type T and supports a sub-array function that returns a portion of the calling objects as a new object of the same type when the starting and ending indices are provided as parameters. The Matrix class is implemented using a MyArray object that holds the row indices and a doubly linked list that contains the column information. The Matrix class also supports a sub-matrix function analogous to that of the MyArray class. As the procedures of the Algorithm

BICLIQUE_MAAPR recur on sub-matrices frequently, the sub-array and sub-matrix functions were proven to be very useful. Three types of partial matrices with unimodal left bound and/or right bound (see Section 2.2.3 for definitions) were derived from the Matrix class. These bounds are implemented using objects of MyArray type and each array element contains a pointer to an element of the doubly linked list that contains the column information.

We assume the vertices of the input convex bipartite graphs are already ordered such that the convex property is satisfied (Otherwise, use Booth and Lueker's algorithm [7] to produce a convex ordering first). Thus, all input graphs are represented using a list of Intervals. In order to produce results that are comparable, all implemented algorithms take a list of Intervals as an input. To verify the correctness of implementations of the algorithms, each algorithm outputs the size of the MECB, $K = (X, Y)$, as well as left endpoint and right endpoint of the X - and Y -side of the K . In case of the B -convex bipartite graph, the identities of the members in X are listed. The overall program outputs the number of edges of each input graph and the time consumption by each algorithm at millisecond precision.

5.1.3 Specific issues related to individual algorithms

The original CHORDAL algorithm [20] takes a Γ -free bipartite adjacency $(0, 1)$ -matrix as an input. As we represent convex bipartite graphs using a list of intervals, the Algorithm CHORDAL was modified accordingly. At the pre-processing stage of the algorithm, elements of the list were sorted by their right endpoints in non-decreasing

order so that the $(0, 1)$ -matrix derived from the list of intervals is a Γ -free bipartite adjacency matrix.

Algorithms BICLIQUE_DOM and BICLIQUE_DOMA are relatively easy to implement. It should be emphasized that the grid defined in the algorithm descriptions is implicit; therefore, no explicit interval to point transformation is needed as long as we know that the left and right end points of an interval correspond to the x - and y - coordinates of a point on the implicit grid. At the pre-processing stage, all intervals are sorted first by left endpoints in non-decreasing order and then by right endpoints in non-increasing order. In BICLIQUE_DOM, the grid is swept from left to right and in each column from top to bottom, while the sorted array is traversed from beginning to end. An auxiliary array is used to store the DOM number of grid points in the current column. An integer variable was used to keep count of input points seen thus far in current column. For BICLIQUE_DOMA, input points instead of grid points are visited in the order dictated by the array. Again an auxiliary array (*y-list*) is used to keep y -coordinates of all points seen thus far in non-increasing order. An integer variable was used to keep track the position of the smallest y in the auxiliary array that is larger than the current x so that insertion of new y can be done efficiently, and the need for growing the array and the need for deleting elements from the end of the array are avoided. The insertion operation into the auxiliary array (*y-list*) makes this algorithm behave like the insertion sort algorithm. This has the following implications: (a) the distribution pattern of the points on the grid will have significant impact on the speed of the algorithm; (b) sorting the

points in non-increasing order of y in the pre-processing stage helps speeding up the algorithm.

Algorithm BICLIQUE_MAAPR is more complex. Conceptually, we model the problem of finding the MECB in a doubly convex bipartite graph as a problem of finding the maximum area axis-parallel rectangle (MAAPR) inscribed in an orthogonally convex rectilinear polygon. Therefore, theoretically speaking, the first step of this algorithm should be transforming the ordered intervals representing a doubly convex bipartite graph into an orthogonally convex rectilinear polygon. However, by properly defining the area function, this step can be omitted as far as implementation is concerned. So, the algorithm directly starts with finding the MAAPR. The MAAPR must be one of the two kinds of rectangles inscribed in a polygon: either 3- or 4-contact rectangles or 2-contact rectangles (Figure 8). All 3- or 4-contact rectangles were examined in linear time with a bottom-up sweep and a top-down sweep. For 2-contact rectangles, finding the MAAPR is essentially a matrix-searching problem. We use each pair of opposing vertex chains to define a double staircase matrix introduced in [9]. The bottom chains (bottom left chain and bottom right chain) define rows and the top chains (top left chain and top right chain) define columns of the matrices. As each row and each column is associated with a vertex, each entry in the area matrices represents the area of the rectangle defined by a pair of opposing vertices. The catch here is that these matrices do not contain explicit values. The entries are computed on the fly as needed using an area function that takes row and column information as parameters. A flag variable with only two possible values (either 1 or -1) is used as another parameter of the area function to indicate the orientation of the

diagonal of the rectangle used to compute its area. Each call to the area function takes constant time. The left and right boundaries of valid entries in the matrices are determined using a top-down sweep. The double staircase matrices were decomposed into a set of rising and reverse rising single staircase matrices as shown in Figure 14. Aggarwal's matrix searching algorithm PartialMaxCompute [1] was then implemented to find the index of the maximum column in each row.

5.2 Experimentation

The goal of the experiments was to evaluate the behaviour of the implemented algorithms given inputs of varying size and distribution patterns. Two major input parameters for the program are number of intervals ($|A|$) and a *top-limit* ($|B|$). Each interval is in the range $(0, \text{top-limit})$. Each combination of sizes of $|A|$ and $|B|$ was run on 6 individually generated graph instances. Average running time was used as a measure of algorithm efficiency.

5.2.1 Input generation

The input to each algorithm was either a convex bipartite graph (not applicable to the BICLIQUE_MAAPR) or doubly convex bipartite graph represented using an array of intervals. For convex bipartite graphs, the intervals are generated randomly. Lengths of intervals follow a normal distribution with $mean = \text{top-limit}/2$, $deviation = mean/4$, $length = \lfloor mean \pm deviation * VAR \rfloor$, where VAR is the standard normal variate. Left endpoints of the intervals are independently and uniformly distributed in the range $(0, \text{top-limit} - length)$. System time was used as a seed to random number generators to ensure that different streams of random numbers were generated each time the input-

generating program was run. For doubly convex bipartite graphs, the order of the intervals is geometrically defined, and thus the intervals are generated in the order of their y-coordinates. All defining points (including bottom-left corner, bottom-right corner, top-left corner, top-right corner, left-extreme and right-extreme) are generated randomly with a uniform distribution, and an illustration of the defining points is given in Figure 8.

5.2.2 Experiment 1: Running time on convex bipartite graphs

The efficiencies of algorithms were evaluated under three different scenarios: (i) $|A|$ and $|B|$ were equal at sizes of $1,000$, $2,000$, $4,000$, $8,000$, $16,000$, and $32,000$; (ii) $|B|$ was fixed at $16,000$ and $|A|$ varied between $1,000$ and $32,000$; (iii) $|A|$ was fixed at $16,000$ and $|B|$ was varied. The results are shown in Table 3 and Figure 11. Running time was presented in seconds (s). In Tables 3 and 4, algorithm names are truncated to fit into the column headings. (Results related to DOMAL – shorthand for algorithm BICLIQUE_DOMAL introduced in later Section 6.3, are discussed in Section 6.3.3.) The *italic* numbers denote the standard errors associated with entries located immediately above them. The entries with *** indicate that the particular algorithm ran out of memory before completion.

The results are summarized as follows:

- 1). Both algorithms BICLIQUE_DOM and BICLIQUE_DOMA were significantly faster than algorithm CHORDAL under all three scenarios. When a critical size of $|B| \times |A|$ ($8,000 \times 8,000$ in first scenario and $4,000 \times 16,000$ in the second and third scenarios) is reached, the performance of CHORDAL deteriorates dramatically. This behaviour is due to the larger space requirement of the algorithm. As $|A|$ and/or $|B|$ increase, $|E|$ gets large

rapidly. Increased virtual memory usage can cause high page-fault rate, resulting in poor performance.

Table 3. Results on convex bipartite graphs.

$ B $	$ A $	Density	Edge	MECB	DOM (s)	DOMA (s)	DOMAL (s)	CHORD AL (s)
1,000	1,000	0.569	568,791 1,450	207,659 1,157	0.078 0.000	0.068 0.003	0.063 0.000	0.159 0.002
2,000	2,000	0.572	2,286,277 3,629	835,492 4,772	0.302 0.006	0.240 0.003	0.188 0.000	0.610 0.000
4,000	4,000	0.571	9,139,530 11,470	3,311,085 14,758	1.141 0.007	0.914 0.003	0.547 0.059	2.867 0.012
8,000	8,000	0.558	35,742,795 737,575	12,832,518 310,946	4.521 0.010	3.320 0.298	2.383 0.032	15.919 1.171
16,000	16,000	0.569	145,776,774 113,782	52,515,833 128,108	17.864 0.039	14.120 0.013	11.570 0.146	*** ***
32,000	32,000	0.570	584,035,795 219,513	210,654,806 200,358	72.700 0.740	57.081 0.349	47.031 0.902	*** ***
16,000	1,000	0.571	9,135,963 32,937	3,359,847 26,749	16.987 0.012	0.063 0.000	0.177 0.003	2.373 0.011
16,000	2,000	0.571	18,271,179 41,463	6,600,741 49,464	17.196 0.062	0.258 0.015	0.552 0.005	4.976 0.026
16,000	4,000	0.568	36,373,037 37,508	13,051,761 58,044	17.448 0.074	0.956 0.031	1.724 0.031	13.338 0.168
16,000	8,000	0.570	72,964,022 69,485	26,478,026 27,879	17.825 0.063	3.615 0.024	5.378 0.110	495.352 48.374
16,000	16,000	0.569	145,776,774 113,782	52,515,833 128,108	17.864 0.039	14.120 0.013	11.570 0.146	*** ***
16,000	32,000	0.570	292,085,624 68,724	105,212,775 166,581	18.419 0.074	56.617 0.126	15.971 0.373	*** ***
1,000	16,000	0.570	9,116,135 3,721	3,278,916 8,819	0.271 0.003	14.258 0.080	0.591 0.005	3.664 0.010
2,000	16,000	0.570	18,241,324 15,048	6,625,116 18,557	0.487 0.003	14.242 0.061	0.792 0.018	7.146 0.048
4,000	16,000	0.570	36,461,672 28,372	13,150,211 32,165	1.341 0.006	14.213 0.021	1.362 0.010	14.399 0.044
8,000	16,000	0.570	72,906,769 60,105	26,279,471 42,820	4.758 0.031	14.287 0.063	3.812 0.035	484.451 16.504
16,000	16,000	0.570	145,989,339 50,215	52,899,540 115,733	17.825 0.015	14.170 0.027	11.084 0.254	*** ***
32,000	16,000	0.569	291,484,689 233,916	104,810,219 247,927	71.531 0.381	14.375 0.069	23.388 0.941	*** ***

Figure 11. Running time in Experiment 1. The top, middle and bottom panels are graphical representations of the running time data in the first, second and third subgroups of Table 3, respectively. In this and following Figures 12 and 13, the numbers on the x-axis denotes size of the inputs ($|A|$ and/or $|B|$), while the numbers on the y-axis mark running time in seconds. Standard error bars are too small to be shown here.

2). As shown in top the panel of Figure 11, when $|A| = |B|$, algorithm BICLIQUE_DOMA consistently outperformed BICLIQUE_DOM at all sizes of $|A|$ and $|B|$.

3). The middle panel of Figure 11 shows that when $|B|$ was fixed at 16,000 while $|A|$ varied from 1,000 to 32,000, running time of BICLIQUE_DOM increased very slowly and consistently from 16.987 ± 0.012 to 18.419 ± 0.074 seconds. This agrees well with the result of our theoretical analysis that BICLIQUE_DOM has a time complexity of $O(|B|^2 + |A|)$. The running time of this algorithm is dominated by $|B|^2$, and the impact of $|A|$ is very small. In this case, running time of BICLIQUE_DOMA increased as a function of $|A|$. However, when $|A| \leq |B|$, BICLIQUE_DOMA is always faster than BICLIQUE_DOM.

4). When $|A|$ was fixed at 16,000 while allowing $|B|$ to vary, the running time of BICLIQUE_DOMA was virtually constant. (Slightly over 14 seconds for all size of $|B|$, see Table 3.) This indicates that BICLIQUE_DOMA's running time is completely independent of $|B|$. This confirms our results of theoretical analysis that the time complexity of BICLIQUE_DOMA is $O(d_{\max}(B) |A|)$. Note that $d_{\max}(B)$ depends on the density of the graph and $|A|$, but not on $|B|$. In case $|B| < |A|$, i.e., $|B| \leq 8,000$, BICLIQUE_DOM outperformed BICLIQUE_DOMA.

5). It should be noted that the performance of BICLIQUE_DOMA is also distribution-dependent. Specifically, when the lengths of intervals has a uniform distribution,

BICLIQUE_DOMA performed better than in the case when intervals are normally distributed, as shown in Figure 12. This is due to the fact that BICLIQUE_DOMA behaves like the insertion sort algorithm. Under normal distribution, intervals with larger left endpoints almost always have larger right endpoints, therefore when the intervals are sorted by their left endpoints and processed in order, insertion of their right endpoints into a decreasing sequence requires traverse of the entire sequence (the auxiliary array of the algorithm). However, if intervals are uniformly distributed, insertion of the right endpoints may only require traverse of half of the decreasing sequence on average. If the right endpoints of intervals have an approximate decreasing order when the left endpoints are sorted in increasing order, then BICLIQUE_DOMA may run in time linear in the number of intervals. This point will be illustrated in the next experiment on doubly convex bipartite graphs. Figure 12 also indicates that BICLIQUE_DOM is independent of distribution of intervals.

Figure 12. Normal distribution vs. uniform distribution. While the running time of BICLIQUE_DOM is independent of distribution of interval lengths, BICLIQUE_DOMA performed better under uniform distribution of interval lengths (DOMA_UNIF) than under normal distribution (DOMA_NORM). See text for explanations.

5.2.3 Experiment 2: Running time on doubly convex bipartite graphs

In addition to the above algorithms, the algorithm BICLIQUE_MAAPR that was specifically designed for doubly convex bipartite graphs was also tested in this experiment under the three scenarios described in Experiment 1. The results are tabulated in Table 4 and graphed in Figure 13.

The results on doubly convex bipartite graphs are summarized as follows:

- 1). Again, all algorithms designed in the present study outperformed CHORDAL under all three scenarios.
- 2). Under all three scenarios, BICLIQUE_DOMA was much faster than BICLIQUE_DOM when $|A| \leq |B|$ as compared to the results on convex bipartite graphs. Moreover, the running time of BICLIQUE_DOMA has larger variance than that observed in Experiment 1. The reason for this is that the geometrical shapes of the orthogonally rectilinear polygon derived from doubly convex bipartite graphs have large impact of the speed of BICLIQUE_DOMA due to its insertion sort-like behaviour. This effect is illustrated in Figure 14. If the polygonal has a diamond shape like the one in Figure 14.a, as the left endpoints increase, the right endpoints decrease; therefore, insertion of a right endpoint into decreasing sequence can be done in nearly constant time. On the contrary, in situation depicted in Figure 14.b, most insertions require a traverse of the entire decreasing sequence (*y-list*).

Table 4. Results on doubly convex bipartite graphs.

$ B $	$ A $	Density	Edge	MECB	MAAPR (s)	DOM (s)	DOMA (s)	DOMAL (s)	CHORDAL (s)
1,000	1,000	0.359	359,139 34,564	184,968 23,434	0.180 0.028	0.076 0.003	0.026 0.008	0.034 0.003	0.120 0.007
2,000	2,000	0.453	1,813,654 217,286	1,116,395 229,590	0.354 0.020	0.284 0.005	0.096 0.019	0.078 0.008	0.534 0.038
4,000	4,000	0.368	5,882,704 654,730	3,416,436 442,427	0.672 0.083	1.120 0.008	0.310 0.080	0.211 0.022	2.164 0.091
8,000	8,000	0.286	18,280,672 4,646,621	8,613,220 3,151,022	1.398 0.201	4.386 0.038	1.279 0.335	0.443 0.063	8.948 1.439
16,000	16,000	0.420	107,590,630 27,255,193	65,586,391 22,672,967	2.966 0.530	17.255 0.118	3.758 1.597	1.456 0.209	*** ***
32,000	32,000	0.466	476,918,396 25,683,514	273,849,236 32,204,621	4.992 1.014	71.560 0.457	15.159 7.700	4.245 0.627	*** ***
16,000	1,000	0.259	4,136,943 1,149,796	2,375,037 822,849	0.133 0.013	16.940 0.023	0.026 0.005	0.044 0.005	1.539 0.197
16,000	2,000	0.353	11,280,736 2,092,026	7,551,243 2,354,048	0.328 0.032	17.320 0.137	0.096 0.023	0.164 0.025	3.774 0.438
16,000	4,000	0.383	24,519,033 5,809,497	16,972,821 5,625,964	0.708 0.058	17.620 0.065	0.227 0.085	0.258 0.026	20.180 11.399
16,000	8,000	0.360	46,135,434 8,271,601	30,332,895 8,307,892	1.320 0.260	17.365 0.135	0.677 0.419	0.500 0.076	289.750 84.944
16,000	16,000	0.420	107,590,630 27,255,193	65,586,391 22,672,967	2.966 0.530	17.255 0.118	3.758 1.597	1.456 0.209	*** ***
16,000	32,000	0.326	166,929,646 26,281,850	69,982,533 18,191,364	6.706 1.097	18.458 0.230	30.263 7.233	4.323 0.274	*** ***
1,000	16,000	0.484	7,739,076 1,646,709	5,289,369 1,388,451	3.143 0.141	0.221 0.018	1.185 0.489	0.547 0.021	3.140 0.486
2,000	16,000	0.639	20,461,861 2,396,176	11,288,218 2,289,016	3.255 0.292	0.448 0.015	3.870 1.009	0.651 0.014	7.664 0.682
4,000	16,000	0.419	26,790,506 3,197,510	10,669,164 1,921,028	4.044 0.644	1.268 0.027	5.790 1.370	0.761 0.048	11.354 0.965
8,000	16,000	0.333	42,584,962 6,993,738	18,715,139 5,869,908	3.636 0.700	4.633 0.062	6.628 2.429	1.070 0.117	245.255 88.320
16,000	16,000	0.285	72,834,887 16,640,156	35,919,718 10,268,168	3.232 0.445	17.612 0.246	5.349 1.079	2.073 0.411	*** ***
32,000	16,000	0.298	152,704,532 27,901,716	87,401,083 24,309,693	2.847 0.226	70.286 0.679	4.240 2.459	1.503 0.412	*** ***

Figure 13. Running time in Experiment 2. Similar to Figure 11, top, middle and bottom panels are graphical representations of the running time data in the first, second and third subgroups of Table 4, respectively. Note the performance of the algorithm BICLIQUE_MAAPR.

Figure 14. Two exemplary orthogonally convex polygons derived from doubly convex bipartite graphs. The x-axis gives the value of the left endpoints and right endpoints of intervals; the y-axis gives the ordering of the intervals. The left sub-chain of each polygon is composed of left endpoints of all intervals, while the right sub-chain consists of their right endpoints. The amounts of time required for BICLIQUE_DOMA to find the MECB in a) and b) are 1.078 and 2.56 seconds, respectively; in contrast, it was 1.922 and 1.562 seconds for BICLIQUE_MAAPR.

3). At smaller size of $|A|$, BICLIQUE_DOMA was faster than BICLIQUE_MAAPR because the latter has a larger overhead than the former. However, as $|A|$ goes beyond 16,000, BICLIQUE_MAAPR becomes much faster than BICLIQUE_DOMA. This is also in agreement with the fact that BICLIQUE_MAAPR is asymptotically faster than BICLIQUE_DOMA (See Table 2).

Chapter 6 Conclusions and directions for future work

The problems of finding the maximum edge cardinality biclique (MECB) in convex bipartite graphs, doubly convex bipartite graphs and bipartite permutation graphs were investigated in this study. Unique structural properties of these graph classes were exploited to devise efficient algorithms for the above problems. The convexity of vertex ordering allows us to represent these graphs using a set of intervals, and solve the MECB problems by manipulating these intervals. As the number of intervals is generally much smaller than the number of edges, the size of problems is greatly reduced as compared to algorithms that must take the number of edges as an input parameter. Although our primary subjects are convex bipartite graphs, the algorithms developed in this study may also be applicable in cases where interval-related problems are primary concerns.

6.1 Conclusions

6.1.1 Convex bipartite graphs

The convex bipartite graph is a subclass of the chordal bipartite graph. Kloks and Kratsch found that for a chordal bipartite graph $G = (A, B, E)$, the number of maximal bicliques in

G is $O(|E|)$ [20]. For a B -convex bipartite graph $G = (A, B, E)$ with $|A| = n$, $|B| = m$, we found that the number of maximal bicliques in G is $O(n^2)$. Although this measure is roughly equivalent to the result by Kloks and Kratsch (as $|E| = O(mn)$), it is independent of m and $|E|$.

By taking advantage of the fact that the neighbourhood of each vertex in A is an interval on B for a convex bipartite $G = (A, B, E)$, we designed two algorithms, BICLIQUE_DOM and BICLIQUE_DOMA, for finding the MECB in convex bipartite graphs. Both algorithms require $O(|A|)$ pre-processing time and are easy to implement. The Algorithm BICLIQUE_DOM uses $O(|A| + |B|)$ storage, and has a time complexity of $O(|B|^2 + |A|)$. The space and time complexity of the Algorithm BICLIQUE_DOMA are $O(|A|)$ and $O(d_{\max}(B)|A|)$, respectively. Note that BICLIQUE_DOMA has some attractive features: 1) With minor modification, it can be used to enumerate all maximal bicliques in a convex bipartite graph. 2) Its time complexity is independent of $|B|$ and $|E|$, thus, in case $|A| \ll |B|$, this is a tremendous advantage. Although it has a worst case time complexity of $O(|A|^2)$, its actual time complexity is $O(d_{\max}(B)|A|)$. In practice, $d_{\max}(B)$ is much smaller than $|A|$ itself. Therefore it will perform well even when $|A| \approx |B|$. Our experimental results shown in Figure 11 and Figure 13 indicate that in the cases where $|A|$ is smaller than $|B|$, such as in the analysis of the gene expression data matrix, where $|B|$ = number of genes, $|A|$ = number of conditions and $|B|$ is typically much larger than $|A|$, this algorithm runs much faster than Kloks and Kratsch's algorithm [20]. In case $|B| \ll |A|$, BICLIQUE_DOM might be a better choice.

6.1.2 Doubly convex bipartite graphs and bipartite permutation graphs

For doubly convex bipartite graphs $G = (A, B, E)$, not only an interval representation is possible, a unique ordering of these intervals exists as well such that G can be depicted using two-dimensional geometrical objects, namely, orthogonally convex rectilinear polygons P composed of two pairs of xy -monotone vertex chains (Figure 8). We have shown that there is a one-to-one mapping between the bicliques in G and axis-parallel rectangles inscribed inside P , and the size of the bicliques equals to the area of the corresponding rectangles. Accordingly, the problem of finding the MECB in G is transformed into the problem of finding the maximum area axis-parallel rectangle (MAAPR) in P . We have shown that this transformation can be done in time $O(|A|)$. Daniels et al. [10,9] provided a solution to the problem of finding the MAAPR in orthogonally convex polygons. Their solution is based on the matrix searching techniques developed by Aggarwal and Klawe [1]. The best known matrix searching algorithm runs in time $O(p\alpha(q) + q)$, where p and q are the number of rows and columns, respectively, and $\alpha(q)$ is a very slowly growing functional inverse of the Ackermann's function. The Algorithm BICLIQUE_MAAPR is designed to find the MECB in doubly convex bipartite graphs following the approach of Daniels et al. In our case, both p and q is $O(|A|)$. Therefore, the overall algorithm runs in time $O(|A|\alpha(|A|))$.

The bipartite permutation graph is a subclass of doubly convex bipartite graph. In addition to the convex property, it has the enclosure property (see Section 2.1). When translated into the language of polygons, this property means that the resulting orthogonally convex rectilinear polygon is composed of only a pair of xy -monotone

vertex chains. Finding the MAAPR in this type of polygons takes linear time. Therefore, the Algorithm BICLIQUE-PERM finds the MECB in bipartite permutation graphs in time $O(|A|)$.

The Algorithms BICLIQUE_DOM, BICLIQUE_DOMA and BICLIQUE-MAAPR are implemented along with algorithm CHORDAL of Kloks and Kratsch [20]. Experimental results obtained on these algorithms substantiated our results of theoretical analysis.

6.1.3 Rules for choosing the best performance algorithms

These algorithms offer several choices for the MECB problem when convex bipartite graphs with various $|A|$ and $|B|$ are encountered:

1). For both convex bipartite graphs and doubly convex bipartite graphs, if $|A|$ and $|B|$ are small ($\leq 8,000$), BICLIQUE_DOMA and BICLIQUE_DOM are preferable due to their efficiency at smaller input size and ease of implementation. If $|A| \leq |B|$, chose BICLIQUE_DOMA. Otherwise, choose BICLIQUE_DOM instead.

2). If $|A|$ and $|B|$ are large ($> 8,000$), for convex bipartite graphs, choose BICLIQUE_DOMA if $|A| \leq |B|$, choose BICLIQUE_DOM otherwise; for doubly convex bipartite graphs, choose BICLIQUE_MAAPR.

3). For bipartite permutation graphs, always choose BICLIQUE_PERM.

4). In terms of microarray data analysis, the number of conditions is always much less than the number of genes. Therefore, if the data matrices are modeled as such that the resulting bipartite graphs are convex on conditions, choose BICLIQUE_DOM, if it is convex on genes, BICLIQUE_DOMA should be considered.

6.2 Directions for future work

The starting point of this study is the biclustering analysis of microarray data matrix. In order to apply our algorithms, a convex bipartite subgraph must be extracted first from a bipartite graph representation of the gene expression data matrix. However, the best known algorithm that produce a convex ordering, if one exists, given a bipartite graph runs in time $O(|E|)$ [7]. Therefore, the vertex-ordering step may represent a major hurdle for high-density graphs. It would be interesting to know if a maximum edge cardinality convex bipartite subgraph can be found in time $o(|E|)$. Upon identification of the MECB in a convex bipartite subgraph, it has to be extended to find a maximal biclique in the original bipartite graph. Currently, the extension process also takes $O(|E|)$ time [3]. The problem of how to extend bicliques found in subgraphs more efficiently has yet to be tackled.

Our bipartite graph model of DNA microarray data matrices only encompasses significant ‘increases’ in gene expression levels (See Table 1 and Section 1.2); the issue of how to incorporate significant ‘decreases’ and ‘no-changes’ into this model remains to be studied.

6.3 Recent developments

Upon completion of design, analysis and experimentation of the above algorithms, we have very recently designed a new algorithm for the problem of finding the MECB in convex bipartite graphs following a different approach. This algorithm is experimentally promising, although its time complexity is not well understood so far. This algorithm is described here solely as an extension of the work completed in previous chapters, and it is not our intention to consider this algorithm as an integral part of the thesis.

Figure 15. A surface view of a portion of a grid. The x - and y -coordinates are left and right endpoints of intervals, and the height of each point on the surface is its S value. The surface is shaded with light source at $(0, 0, 1)$.

6.3.1 A bird-view of the grid

When the $n \times n$ grid was inspected column by column, it was noticed that given a column x , the distribution of $S(x, y)$ in x is multi-modal, namely, there are a number of local

maxima in the sequence $\{ S(x, y_j): 1 \leq j \leq n \}$. The column local maxima, denoted by CLM, tend to persist in adjacent columns, i.e., if $S(x_i, y_{j-1}) \leq S(x_i, y_j) \geq S(x_i, y_{j+1})$, then the following is probably also true: $S(x_{i+1}, y_{j-1}) \leq S(x_{i+1}, y_j) \geq S(x_{i+1}, y_{j+1})$. When a stretch of columns is examined, these CLMs will form ridges in a number of rows. Similar structures exist in the adjacent rows (Figure 15). Let's denote row local maxima by RLM. Thus, when the entire grid is envisioned with only local maxima in mind, a higher level of grid-like structure depicted in Figure 16 appears.

Figure 16. A schematic view of the ridges composed of column local maxima and row local maxima.

This observation prompted us to consider if it is possible to only keep tracks of the local maxima when the original grid is swept column-wise. The rationale behind this idea is that, if we can exclude substantial amount of points from computation by only considering local maxima, we may improve on algorithm BICLIQUE_DOMA, which examines almost all points on the original $n \times n$ grid. Before finding a way of tracking local maxima, we must find how they are formed and how they evolve when we move from left to right column-wise.

Figure 17. Evolution of CLM. If N_2 is a column local maximum (CLM), M_2 is likely to be a CLM too. But addition of input points and changing x values makes it hard to predict.

In Figure 17, define the point (x_0, y_0) be the point just outside the top-left corner of the grid. Let N_{j-1} be the number of input points in the rectangular region defined by $(x_0, x_i]$ and $(y_0, y_{j-1}]$, N_j be the number of input points in the rectangular region defined by $(x_0, x_i]$ and $(y_{j-1}, y_j]$, N_{j+1} be the number of input points in the rectangular region defined by $(x_0, x_i]$ and $(y_j, y_{j+1}]$, M_{j-1} be the number of input points in the rectangular region defined by $(x_i, x_{i+1}]$ and $(y_0, y_{j-1}]$, M_j be the number of input points in the rectangular region defined by $(x_i, x_{i+1}]$ and $(y_{j-1}, y_j]$, M_{j+1} be the number of input points in the rectangular region defined by $(x_i, x_{i+1}]$ and $(y_j, y_{j+1}]$. Let $D_1 = y_{j-1} - y_j$, $D_2 = y_j - y_{j+1}$, and $D_3 = x_{i+1} - x_i$. Then:

$$S(x_i, y_{j-1}) = N_{j-1} \times (y_{j-1} - x_i).$$

$$S(x_i, y_j) = (N_{j-1} + N_j) \times (y_j - x_i).$$

$$S(x_i, y_{j+1}) = (N_{j-1} + N_j + N_{j+1}) \times (y_{j+1} - x_i).$$

$$S(x_{i+1}, y_{j-1}) = (N_{j-1} + M_{j-1}) \times (y_{j-1} - x_{i+1}).$$

$$S(x_{i+1}, y_j) = (N_{j-1} + N_j + M_{j-1} + M_j) \times (y_j - x_{i+1}).$$

$$S(x_{i+1}, y_{j+1}) = (N_{j-1} + N_j + N_{j+1} + M_{j-1} + M_j + M_{j+1}) \times (y_{j+1} - x_{i+1}).$$

Assume $S(x_i, y_j)$ is a CLM, namely, $S(x_i, y_{j-1}) \leq S(x_i, y_j) \leq S(x_i, y_{j+1})$, then we have:

$$(N_{j-1} + N_j) \times (y_j - x_i) - (N_{j-1} + N_j + N_{j+1}) \times (y_{j+1} - x_i) \quad (1)$$

$$(N_{j-1} + N_j) \times (y_j - x_i) - N_{j-1} \times (y_{j-1} - x_i) > 0 \quad (2)$$

From (1), we have

$$D_2 \times (N_{j-1} + N_j + N_{j+1}) - N_{j+1} \times (y_j - x_i) > 0 \quad (3)$$

From (2), we have

$$N_j \times (y_j - x_i) - N_{j-1} \times D_1 > 0 \quad (4)$$

We are interested in finding out if $S(x_{i+1}, y_j)$ is also a CLM. First, let us examine under what conditions, we will have $S(x_{i+1}, y_j) > S(x_{i+1}, y_{j+1})$:

$$\begin{aligned} & S(x_{i+1}, y_j) - S(x_{i+1}, y_{j+1}) \\ &= (N_{j-1} + N_j + M_{j-1} + M_j) \times (y_j - x_{i+1}) - (N_{j-1} + N_j + N_{j+1} + M_{j-1} + M_j + M_{j+1}) \times (y_{j+1} - x_{i+1}) \\ &= [D_2 \times (N_{j-1} + N_j + N_{j+1}) - N_{j+1} \times (y_j - x_i)] + [D_2 \times (M_{j-1} + M_j + M_{j+1}) + D_3 \times (N_{j+1} + M_{j+1}) - M_{j+1} \times (y_j - x_i)]. \end{aligned}$$

From (3) we know the first part of the above expression is non-negative, and the second part of it will be non-negative also if $M_{j+1} = 0$. This means that we will have $S(x_{i+1}, y_j) > S(x_{i+1}, y_{j+1})$ if there is no input point at (x_{i+1}, y_{j+1}) . In other words, existence of input points below a CLM suggests possibility for shifting the current one to lower point or producing a new CLM.

Next, let us examine under what conditions, we will have $S(x_{i+1}, y_j) > S(x_{i+1}, y_{j-1})$:

$$\begin{aligned}
& S(x_{i+1}, y_j) - S(x_{i+1}, y_{j-1}) \\
&= (N_{j-1} + N_j + M_{j-1} + M_j) \times (y_j - x_{i+1}) - (N_{j-1} + M_{j-1}) \times (y_{j-1} - x_{i+1}) \\
&= (N_j + M_j) \times (y_j - x_i - D_3) - (N_{j-1} + M_{j-1}) \times D_1
\end{aligned}$$

Figure 18. A graphical interpretation of the dynamics of CLMs. A. Addition of two peaks that are not aligned may produce a peak at a new position. B. The impact of the new profile by input points is long lasting if the existing profile has a series of increasing peaks.

If we treat $(y_j - x_i)$ as a given constant, the above expression clearly will have a negative outcome if D_1 , D_3 , N_{j-1} or M_{j-1} is sufficiently large with respect to $(N_j + M_j)$. Conversely, if $(N_j + M_j) \times (y_j - x_i)$ is the dominant term, then $S(x_{i+1}, y_j)$ will continue to be a CLM. In other words, as we move from left to right, current CLM may remain unchanged or be shifted up, or new CLM may be generated if one of the four parameters (D_1 , D_3 , N_{j-1} or M_{j-1}) is sufficiently large. Of the four parameters, only M_{j-1} is associated with addition of new input points. Therefore, it should be emphasized that new CLM can be formed even

without addition of new input points to the column, and the position of the new CLM may not necessarily coincide with an input point. Given a new CLM, if it coincides with an input point, we call it input-related CLM, otherwise, we call it non-input-related CLM. Above analysis shows that an existing CLM can be preserved in the next column or can be shifted up or down; new CLM can be formed either by addition of input points or by moving from left to right (Figure 18). Therefore, in order to keep track of CLMs, we must have means for detecting new CLMs as we sweep column-wise from left to right. Input-related new CLMs are easy to find, as we know the exact positions of input points. However, non-input-related new CLMs are not easy to detect.

Intuitively, a global maximum must be a CLM and a RLM at the same time. Like CLMs, RLMs also tend to persist in subsequent rows going downwards. We argued that if a global maximum was formed at a point $p_{max} = (x_{max}, y_{max})$, then the CLM immediately below it must be a RLM also. In other words, if a CLM is also a RLM, we should trace it upwards to see if it will lead to a non-input-related new CLM. This way, the global maximum will be located eventually.

6.3.2 Algorithm BICLIQUE_DOMAL

An algorithm called BICLIQUE_DOMAL was designed subsequently based on the above idea. The algorithm still uses a plane-sweep approach. However unlike the BICLIQUE_DOMA that examines each maximal point at each column, algorithm BICLIQUE_DOMAL only examines CLMs and input points. Specifically, at the pre-processing stage, a layered range tree was built on the set of input points to answer the dominance counting query. A doubly linked list L_{max} sorted on y -coordinates in

descending order was used to store CLMs. At each column, the following five tasks were performed in order. **a)** From top to bottom, compute S for each input point. If an input point is a CLM, insert it into a temporary list $Temp_{max}$. **b)** Update L_{max} , which initially contains CLMs of the previous column. For each CLM in L_{max} update its S to incorporate the change in x -coordinate. If it is moved up or down, update its y -coordinates also. If it is not a CLM anymore, remove it from L_{max} . **c)** Merge L_{max} and $Temp_{max}$. **d)** Detect non-input-related CLMs. For each CLM in L_{max} , if it is a RLM as well, trace it up until it is not a RLM anymore or another CLM in L_{max} is reached. If a new CLM with higher S value than that of the current CLM is detected, insert it into L_{max} . **e)** From L_{max} , remove any CLMs whose S value is smaller than those of the CLMs above and below it (See Figure 19). According to above analysis, it is impossible for these CLMs to be the global maximum in next column. This step helps to further reduce the number of CLMs to be tracked.

Figure 19. Weeding of insignificant peaks. Each peak represents a CLM. Only A, B, C, D, E are kept, the smaller ones in between are removed. The CLMs are laid from left to right instead of from top to bottom in order to view peaks.

ALGORITHM BICLIQUE_DOMAL (I_A)

Input: a convex bipartite graph $G = (A, B, E)$. Assume G is represented with a list of intervals I_A .

Output: the maximum edge cardinality biclique $K = (X, Y)$ and the size of K .

1. Transform intervals in I_A into 2-dimensional points with left endpoints and right endpoints as x - and y -coordinates, respectively; and build a layered range tree on these points. Assume there are n_x x -coordinates and n_y y -coordinates. Make an empty list L_{max} to hold column local maxima (CLM).

2. At each column x in I to n_x :

2.1 Starting from the first input point in x , for each y value, if $S(x, y)$ is a column local maximum, insert $CLM(x, y)$ into temporary list $Temp_{max}$.

2.2 If L_{max} is non-empty, update each CLM basing on x .

2.3 Merge L_{max} and $Temp_{max}$.

2.4 Remove $CLMs$ that are smaller than $CLMs$ both above and below it.

2.5 For each CLM in L_{max} , if it is also a RLM , do

Define current CLM as CLM_{cur} , and define the CLM immediately above CLM_{cur} as CLM_{pre} .

If there is no input point between CLM_{cur} and CLM_{pre} , then

Process_Without_Input(CLM_{cur} , CLM_{pre}),

Else

Process_With_Input(CLM_{cur} , CLM_{pre}),

3. Perform a linear scan of the list I_A to identify X . $K = |X| \times |Y|$.

END BICLIQUE_DOMAL

Process_Without_Input(CLM_{cur} , CLM_{pre})

Starting from CLM_{cur} moving upwards, do

Determine if each point (x, y) encountered is a *CLM*.

If a new *CLM* (x, y) is identified and $S(x, y) > S(x_{cur}, y_{cur})$, then

Insert *CLM* (x, y) into L_{max} .

Stop if a point is not a *RLM* anymore, or if $S(x, y) < S(x_{pre}, y_{pre})$.

If $S(x, y)$ is larger than the current global maximum, $Y = y - x$.

END Process_Without_Input

Process_With_Input(CLM_{cur}, CLM_{pre})

Starting from CLM_{cur} moving upwards, do

Determine if each point (x, y) encountered is a *CLM*.

If a new *CLM* (x, y) is identified and $S(x, y) > S(x_{cur}, y_{cur})$, then

Insert *CLM* (x, y) into L_{max} .

Stop if a point is not a *RLM* anymore, or if CLM_{pre} is reached.

If $S(x, y)$ is larger than the current global maximum, $Y = y - x$.

END Process_With_Input

Note the only difference between Process_With_Input and Process_Without_Input is the stop condition.

6.3.3 Analysis of complexity

A tentative analysis of the time complexity is given as follows. In step 1, transformation from intervals into points takes $O(n)$ time. Construction of a layered range tree of n input points requires $O(n \log n)$ time. Also, the input points are sorted by x - and y -coordinates. Associated with each x (or y) is a binary search tree that stores y (or x) values of all points

sharing the same x (or y) coordinate. Sorting all x (or y) require $O(n)$ time and building binary search trees using an array of sorted x (or y) values takes $O(n)$ time as well. In step 2.1, for each input point visited, we need to determine if it is a CLM. Once the S value of an input point p_{in} is computed through querying the layered range tree, S values of the four points around p_{in} can be determined in time $O(\log k)$. Step 2.2 – 2.4 is proportional to the average length of the L_{max} . Step 2.5 is the most time consuming step. However, it is performed only if the CLM is also a RLM. Its time consumption is further limited by only visiting the portions of the column that contain input points. Altogether, step 2 visits $O(h)$ points and $O(\log k)$ time is spent at each points, thus $O(h \log k)$ time is required. The number k is usually insignificant, as $k = n/n_x$, where n_x is the number of unique x 's. On average, h should be sub-quadratic, but its worst-case complexity cannot be established at this moment. Step 3 takes $O(n)$ time apparently. Therefore, the total time complexity is $O(n \log n + h \log k)$. The space requirement is $O(n \log n)$ due to the layered range tree.

The following theorem summarizes the results of above analysis.

Theorem 6.5.3: *The Algorithm BICLIQUE_DOMAL finds the MECB in a convex bipartite graph in time $O(n \log n + h \log k)$, where h is the number of points visited on the grid and k is the average number of points in each column and row. The algorithm uses $O(n \log n)$ space.*

Experimental results on this algorithm can be found in Tables 3 and 4 (listed under DOMAL), as well as in Figures 11 and 13. Although the upper bound of this algorithm is

not solidly established due to lack of knowledge on the number of points h visited in average case, this algorithm is experimentally very promising. It outperformed both BICLIQUE_DOM and BICLIQUE_DOMA when $|A| = |B|$ and when $|B| = 16,000$ with $|A|$ varying. However, when $|A|$ is fixed at $16,000$, it behaved like BICLIQUE_DOM with a much slower growth rate though. This observation suggests that h is dependent on $|B|$, which defines the grid size when $|B| < |A|$. In case of doubly convex bipartite graphs, BICLIQUE_DOMAL behaved like BICLIQUE_MAAPR, which has a time complexity of $O(|A|\log\log|A|)$.

This algorithm offers yet another option for the MECB problem in terms of choice of algorithms: When convex bipartite graphs with $|A| = |B|$ are encountered, BICLIQUE_DOMAL should always be used in place of BICLIQUE_DOMA when $|A|$ and $|B|$ are large ($> 8,000$, see Figure 11).

Bibliography

- [1] Aggarwal, A., Klawe, M. "Application of generalized matrix searching to geometric algorithms", *Discrete Applied Mathematics*, 1990, 27: 3-23.
- [2] Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilber, R. "Geometric applications of a matrix-searching algorithm", *Algorithmica*, 1987, 2: 195-208.
- [3] Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P. L., Simeone, B. "Consensus algorithms for the generation of all maximal bicliques", *DIMACS Technical Report 2002 - 52*, 2002.
- [4] Amilhastre, J., Janssen, P., Vilarem, M.-C. "Computing a minimum biclique cover is polynomial for bipartite domino-free graphs", *Proceedings of the eighth annual ACM-SIAM symposium on discrete algorithms*, pp.36-42, ACM, 1997, New Orleans, LA.
- [5] Asratian, A. S., Denley, T. M. J., Häggkvist, R. *Bipartite Graphs and Their Applications*, (1 Ed.), Cambridge University Press, 1998.
- [6] Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z. "Discovering local structure in gene expression data: the order-preserving submatrix problem", *Journal of Computational Biology*, 2003, 10: 373-384.
- [7] Booth, K., Lueker, G. "Testing for consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms", *Journal of Computer and System Sciences*, 1976, 13: 335-379.
- [8] Chen, Y., Church, G. "Biclustering of expression data", *Proceedings of ISBM'00*, pp.93-103, AAAI Press, 2000, San Diego, CA.

- [9] Daniels, K., Milenkovic, V., Roth, D. "Finding the largest rectangle in several classes of polygons", *Technical Report 22--95, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University*, 1995.
- [10] Daniels, K., Milenkovic, V., Roth, D. "Finding the largest area axis-parallel rectangle in a polygon", *Computational Geometry: Theory and Applications*, 1997, 7: 125-148.
- [11] Dawande, M., Keskinocak, P., Swaminathan, J. M. "On bipartite and multipartite clique problems", *Journal of Algorithms*, 2001, 41: 388-403.
- [12] Diestel, R. *Graph Theory*, (2 Ed.), Springer, 2000.
- [13] Eppstein, D. "Aboricity and bipartite subgraph listing algorithms", *Information Processing Letters*, 1994, 51: 207-211.
- [14] Eppstein, D. "SMAWK totally monotone matrix searching algorithm", 2002, <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/117244>
- [15] Getz, G., Levine, E., Domany, E. "Coupled two-way clustering analysis of gene microarray data", *Proceedings of National Academy of Science*, 2000, 97: 12079-12084.
- [16] Glover, F. "Maximum matching in a convex bipartite graph", *Naval Research Logistics Quarterly*, 1967, 4: 313-316.
- [17] Güting, R.-H., Nurmi, O., Ottmann, T. "Fast algorithms for direct enclosures and direct dominances", *Journal of Algorithms*, 1989, 10: 170-186.
- [18] Hochbaum, D. S. "Approximating clique and biclique problem", *Journal of Algorithms*, 1998, 29: 174-200.

- [19] Klawe, M., Kleitman, J. D. "An almost linear time algorithm for generalized matrix searching", *SIAM Journal of Discrete Mathematics*, 1990, 3: 81-97.
- [20] Kloks, T., Kratsch, D. "Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph", *Information Processing Letters*, 1995, 55: 11-16.
- [21] Kluger, Y., Basri, R., Chang, J., Gerstein, M. "Spectral biclustering of microarray data: coclustering genes and conditions", *Genome Research*, 2003, 13: 703-716.
- [22] Lai, T.-H., Wei, S.-S. "Bipartite permutation graphs with application to the minimum buffer size problem", *Discrete Applied Mathematics*, 1997, 74: 33-55.
- [23] Lazzeroni, L., Owen, A. "Plaid models for gene expression data", *Statistica Sinica*, 2002, 12: 61-86.
- [24] Lipski, W. Jr., Preparata, F. P. "Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems", *Acta Informatica*, 1981, 15: 329-346.
- [25] Lubiw, A. "Double lexical ordering of matrices", *SIAM Journal on Computing*, 1987, 16: 854-879.
- [26] Lueker, G. "A data structure for orthogonal range queries", *Proceedings of the 19th Annual IEEE Symposium of Foundation of Computer Science*, pp.28-34, 1978.
- [27] Overmars, M., Wood, D. "On rectangular visibility", *Journal of Algorithms*, 1988, 9: 372-390.
- [28] Peeters, R. "The maximum edge biclique problem is NP-complete", 2000, <http://citeseer.nj.nec.com/peeters00maximum.html>

- [29] Pichat, E. "The disengagement algorithm or a new generalization of the exclusion algorithm", *Discrete Mathematics*, 1977, 17: 95-106.
- [30] Preparata, F. P., Shamos, M. I. *Computational Geometry: An Introduction*, Springer-Verlag, 1985.
- [31] Prisner, E. "Bicliques in graphs I: Bounds on their numbers", *Combinatorica*, 2000, 20: 109-117.
- [32] Schena, M., Shalon, D., Davis, R., Brown, P. "Quantitative monitoring of gene expression patterns with a complimentary DNA microarray", *Science*, 1995, 270: 467-470.
- [33] Spinrad, J. "Doubly lexical ordering of dense 0-1 matrices", *Information Processing Letters*, 1993, 45: 229-235.
- [34] Spinrad, J., Brandstädt, A., Stewart, L. "Bipartite permutation graphs", *Discrete Applied Mathematics*, 1987, 18: 279-292.
- [35] Tanay, A., Sharan, R., Shamir, R. "Discovering statistically significant biclusters in gene expression data", *Bioinformatics*, 2002, 18 Suppl. 1: S136-S144.
- [36] Uehara, R. "Linear time algorithms on chordal bipartite and strongly chordal graphs", *Lecture Notes in Computer Science (Electronic Edition)*, 2002, 2380: 993-1004.
- [37] Willard, D. E. "New data structures for orthogonal range queries", *SIAM Journal of Computing*, 1985, 14: 232-253.
- [38] Yang, J., Wang, H., Wang, W., Yu, P. "Enhanced biclustering on expression data", *Third IEEE Symposium on BioInformatics and BioEngineering (BIBE'03)*, pp.321-321, 2003, Bethesda, MD.

- [39] Yu, C.-W., Chen, G. H. "Efficient parallel algorithms for doubly convex-bipartite graphs", *Theoretical Computer Science*, 1995, 147: 249-265.