COMPUTATIONAL STUDY ON BIDIMENSIONALITY THEORY BASED ALGORITHMS

by

Chunhao Wang

B.Eng., Zhejiang University, 2009

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2011

APPROVAL

Name: Chunhao Wang

Degree: Master of Science

Title of Thesis: Computational Study on Bidimensionality Theory Based Algorithms

Examining Committee: Dr. Pavol Hell
Chair

Dr. Qian-Ping Gu,
Professor, School of Computing Science
Simon Fraser University
Senior Supervisor

Dr. Arthur L. Liestman,
Professor, School of Computing Science
Simon Fraser University
Supervisor

Dr. Binay Bhattacharya,
Professor, School of Computing Science
Simon Fraser University
SFU Examiner

Date Approved: 9 August 2011

Declaration of
Partial Copyright Licence

Abstract

Bidimensionality theory provides a general framework for developing subexponential fixed parameter algorithms for NP-hard problems. A representative example of such algorithms is the one for the longest path problem in planar graphs. The largest grid minor and the branch-decomposition of a graph play an important role in bidimensionality theory. The best known approximation algorithm for computing the largest grid minor of a planar graph has the approximation ratio 3. In this thesis, we report a computational study on a branch-decomposition based algorithm for the longest path problem in planar graphs. We implement the 3-approximation algorithm for computing large grid minors. We also design and implement an exact algorithm for computing the largest cylinder minor in planar graphs to evaluate the practical performance of the 3-approximation algorithm. The results show that the bidimensional framework is practical for the longest path problem in planar graphs.

Keywords: Computational study, bidimensionality theory, branch-decomposition, grid minor, longest path problem

To my parents

"True glory consists in doing what deserves to be written; in writing what deserves to be read."

— Pliny The Elder *(23 AD - 79 AD)*

"Go forward, and faith will follow."

— Jean le Rond d'Alembert *(1717 - 1783)*

Acknowledgments

Expectation, passion and creation: they are all here, as a graduate student at the School of Computing Science, Simon Fraser University. This thesis is a conclusion of the unforgettable two years' life for research and study.

There are many people to whom I owe great gratitude for the existence of this thesis. Foremost, I would like to express my sincere appreciation to my senior supervisor, Dr. Qian-Ping Gu, without whom, my life would be totally different after the year of 2009, let alone the finale of this work. Dr. Gu introduced me into the area of algorithmic graph theory. His encouraging guidance, sharp scientific ideas, insightful thoughts and easy-going character have offered me numerous benefits during my research. For me, he is a perfect exemplar. I wish one day I could be a scientist like him. I am touched by Dr. Gu's help for revising this thesis. He spent hours with me to explain the comments he provided literally for my writing. In addition, I am very grateful to Dr. Gu for the financial support, without which, my living in Vancouver would not be so easy.

The excellent faculty members initiate the academic atmosphere in the school and are very helpful for my research. Taking this opportunity, I would like to thank my supervisor, Dr. Arthur L. Liestman for his advantageous talks during my graduate journey, and for his help for improving my writing skills. I am grateful to the examiner, Dr. Binay Bhattacharya for reviewing my thesis and serving in my defence committee. I also had the pleasure of taking his course, from which I have learned a lot about network location theory. Dr. Liestman and Dr. Bhattacharya's valuable comments and suggestions on my work are very precious presents for me, which make this thesis much better. Many thanks to Dr. Pavol Hell for his time to chair my defence. Moreover, it was my honor to serve as a teaching assistant in his course. Besides, I would like to express my gratitude to (in alphabetical order) Dr. Binay Bhattacharya, Dr. Andrei Bulatov, Dr. Funda Ergun and Dr. Anoop Sarkar

for the knowledge I learned and nice time I had when taking their courses.

I am obliged to Marjan Marzban for her help with the implementation of the algorithms and providing me the dataset for experiments. I am thankful that the technical support staff Johnny Zhang provided me with an emergency laptop which made my Thesis Defence smooth. There are many amigos who share with me their knowledge, thoughts and pleasure. I truly appreciate everything they impart to me. If one does not find his/her name directly in the text, it is between the lines.

Last, I leave the deepest and warmest part of my heart for my beloved parents, who gave birth to me, fed me, enlightened me and educated me with their unconditional support and continuous love. Without them, everything is impossible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

For NP-hard problems, fixed parameter exact algorithms have been extensively studied. Bidimensionality theory developed by Demaine et al. [13, 16, 17, 21] provides a general framework for designing subexponential fixed parameter algorithms for NP-hard problems in a graph G. Readers may refer to two good surveys on bidimensionality theory based algorithms [18, 24]. Bidimensionality theory is based on the relationship between branchwidth and the size of the largest grid minor of G.

The notions of *branchwidth* and *branch-decomposition* are introduced by Robertson and Seymour [45]. Informally, a branch-decomposition of a graph G is a collection of vertex-cut sets represented as links of a tree whose leaves are edges of G. The width of a branch-decomposition is the maximum size of a cut set in the collection and the branchwidth of G, denoted by $\mathrm{bw}(G)$, is the minimum width of all possible branch-decompositions of G. Formal definitions of branch-decomposition and branchwidth are given in Chapter 2. A graph H is called a *minor* of a graph G if H can be obtained from a subgraph of G through a sequence (maybe empty) of edge contractions. A $(g \times h)$-grid is a graph on vertex set $\{(i,j)|0 \leq i < g, 0 \leq j < h, i, j : \text{ integer}\}$ such that vertices (i,j) and (i',j') are adjacent if and only if $|i-i'| + |j-j'| = 1$ (see Figure 1.1 (a) for example). We denote by $\mathrm{gm}(G)$ the largest integer g such that G contains a $(g \times g)$-grid as a minor. Robertson, Seymour and Thomas show that $\mathrm{gm}(G) \leq \mathrm{bw}(G) \leq 20^{2\mathrm{gm}(G)(\mathrm{gm}(G)+1)^4}$ for general graphs [44] and $\mathrm{gm}(G) \leq \mathrm{bw}(G) \leq 4\mathrm{gm}(G)$ for planar graphs [44]. The linear bound of the form $\mathrm{gm}(G) \leq \mathrm{bw}(G) \leq c\mathrm{gm}(G) + o(\mathrm{gm}(G))$ has been extended to bounded genus graphs [14] and to graphs excluding a fixed minor [19].

1.1 Bidimensionality theory based algorithms

Informally, a problem in a graph G with solution value $\chi(G)$ is bidimensional if the value χ for the $(g \times g)$-grid grows as g increases and $\chi(H) \leq \chi(G)$ if H is a minor of G. In the bidimensionality theory framework, to decide if $\chi(G) \geq k$ for given G and k, one first computes or estimates $\mathrm{bw}(G)$. If $\mathrm{bw}(G)$ is large (at least some threshold value) then by the linear bound $\mathrm{bw}(G) \leq c\mathrm{gm}(G)$, $\chi(G) \geq k$ may be concluded. Otherwise, $\chi(G)$ is computed exactly by a branch-decomposition based algorithm which typically runs in polynomial time in the size of G but in exponential time in $\mathrm{bw}(G)$. With a small $\mathrm{bw}(G)$, $\chi(G)$ can often be computed efficiently.

A representative example of bidimensional problems is the longest path problem. Given a graph G and an integer k, the problem is to determine if there is a path of length at least k in G. The optimization version of the problem is to find a path with the largest length in G. Although the longest path problem can be solved in polynomial time for some classes of graphs such as interval graphs [42], circular-arc graphs [10], biconvex graphs [1], and cocomparability graphs [11], it is NP-hard for general graphs and remains NP-hard for many classes of graphs [9, 29, 36, 40, 41, 50], including planar graphs [28]. The problem is a generalization of the Hamiltonian path problem and has many applications such as circuit design [35]. Because of its importance, the longest path problem has been extensively studied and bidimensionality theory based algorithms have been developed for the problem in planar graphs [25], graphs with bounded genus, and graphs excluding a fixed minor [23].

Although bidimensionality theory based algorithms have been proposed for many NP-hard problems [18, 24], little is known about the practical performance of these algorithms. Hurdles for the computational studies of these algorithms may include the implementations of algorithms for computing $\mathrm{bw}(G)$, the branch-decomposition based algorithms for exactly solving the problems and algorithms for finding the grid minors of size guaranteed by the linear bound $\mathrm{bw}(G) \leq c\mathrm{gm}(G)$. None of these implementations are trivial. Recently, progress has been made for computing the branchwidth and branch-decomposition efficiently in large planar graphs [3, 4]. In this thesis, we implement the algorithm for computing large grid minors and the branch-decomposition based algorithm for solving the planar longest path problem. We use these tools to perform a computational study on the bidimensionality theory based algorithms for the longest path problem in planar graphs.

Let G be a planar graph and β an integer. It is known that $\mathrm{bw}(G) \leq \beta$ can be decided

in $O(n^2)$ time [48] and a branch-decomposition of width $\mathrm{bw}(G)$ can be computed in $O(n^3)$ time [33, 48]. Based on the sphere-cut decomposition (a branch-decomposition with a certain geometric property) and the non-crossing property of planar graphs embedded on a sphere, Dorn, Penninkxa, Bodlaender and Fomin give a branch-decomposition based approach which solves Hamiltonian path like problems in $2^{O(\mathrm{bw}(G))}n^{O(1)}$ time [25]. Especially, they show that the planar longest path problem can be solved in $O(2^{3.404\mathrm{bw}(G)}n^{O(1)} + n^3)$ time [25]. We refer to Dorn et al.'s algorithm for solving the planar longest path problem as the *DPBF Algorithm*. Efficient algorithms for computing grid minors with the size guaranteed by the linear bound $\mathrm{bw}(G) \leq c\mathrm{gm}(G)$ are also known [5, 32]. Let $\chi(G)$ denote the length of the longest path of G. To decide if $\chi(G) \geq k$, $\mathrm{bw}(G)$ is first computed. If $\mathrm{bw}(G) \geq c\sqrt{k+1}$ then $\chi(G) \geq k$ is concluded. Otherwise, $\chi(G)$ is computed by the DPBF Algorithm in $O(2^{3.404\mathrm{bw}(G)}n^{O(1)} + n^3) = O(2^{3.404c\sqrt{k}}n^{O(1)} + n^3)$ time. Since the coefficient c in the linear bound appears in the exponent of the running time of bidimensionality theory based algorithms, much work has been done to reduce this coefficient [14, 15, 17, 20]. Very recently, Gu and Tamaki have reduced c to 3, that is, $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ [32], which implies that $\chi(G)$ can be computed in $O(2^{10.212\sqrt{k}}n^{O(1)} + n^3)$ time.

A $(g \times h)$-cylinder is a graph on vertex set $\{(i,j)|0 \leq i < g, 0 \leq j < h, i, j : \text{ integer}\}$ such that vertices (i,j) and (i',j') are adjacent if and only if $i' \equiv (i \pm 1) \bmod g$ and $j' = j$ or $i' = i$ and $|j - j'| = 1$ (see Figure 1.1 (b) for example). Notice that a $(g \times h)$-cylinder contains a $(g \times h)$-grid as a minor. Let $\mathrm{cm}(G)$ denote the largest integer g such that G contains a $(g \times \lceil g/2 \rceil)$-cylinder as a minor. Gu and Tamaki show that $\mathrm{bw}(G) \leq 2\mathrm{cm}(G)$ for a planar graph G and their algorithm, which we refer to as the *GT Algorithm*, actually computes $(g \times h)$-cylinder minors of G [32]. Using the cylinder minor instead of the grid minor of G, we can use $\mathrm{bw}(G) \geq 2\sqrt{2(k+1)}$ to conclude $\chi(G) \geq k$. When $\mathrm{bw}(G) < 2\sqrt{2(k+1)}$, we compute $\chi(G)$ in $O(2^{9.628\sqrt{k}}n^{O(1)} + n^3)$ time.

1.2 Contributions

We develop and implement an exact algorithm for computing the largest cylinder minor of a planar graph. The algorithm is based on the enumeration of all cycles of a certain length between two faces of a planar graph. There is no known polynomial bound on the number of cycles of a certain length. Therefore the algorithm may run in exponential time. The computational results show that the implementation of this exact algorithm can handle small

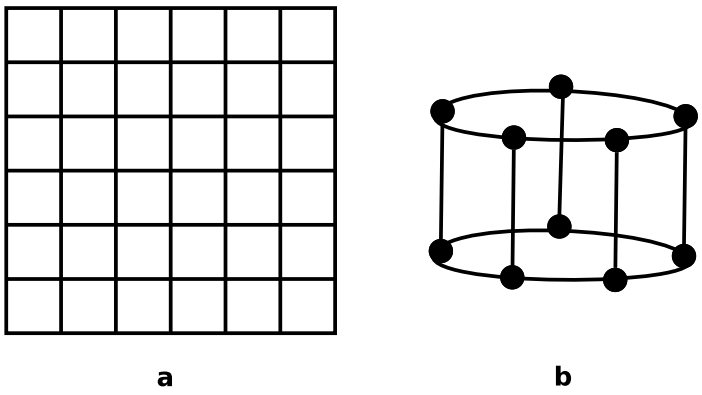

Figure 1.1: An example of grid and cylinder. A (7×7)-grid is shown in (a) and a (5×2)-cylinder is shown in (b).

instances of graphs of less than 100 vertices with 2 GByte memory. The exact algorithm provides a tool to evaluate faster approximation algorithms for computing large grid minors.

To perform the computational study for the planar longest path problem, we implement the DPBF Algorithm [25] for computing the longest path and the GT Algorithm [32] for computing large grid minors in a planar graph G. We use the implementation reported by Bian et al. [4] of the $O(n^2)$ time algorithm [48] for computing $\mathrm{bw}(G)$ and the implementation by Bian and Gu [3] of the $O(n^3)$ time algorithm [3, 48] for computing an optimal branch-decomposition.

We test the performance of the GT Algorithm for computing cylinder minors and the DPBF Algorithm for the longest path on a wide range of planar graphs with up to 15,000 edges. The computational results show that the GT Algorithm is practical and the cylinder minors it finds are close to optimal. The DPBF Algorithm is practical for large planar graphs when the branchwidth is not greater than 10.

To our best knowledge, this work is the first computational study on bidimensionality theory based algorithms. We implement these algorithms and show that this framework is practical for large instances of planar graphs with thousands of edges. Moreover, we provide a tool — the exact algorithm for computing the largest cylinder minor in planar graphs — to evaluate the performance of the approximation algorithms for computing large grid minors. Although our implementation of the exact algorithm could handle only small instances of planar graphs, the computational results show that the cylinder minors found

by the GT Algorithm are very close to the optimal ones. Therefore we could expect that for large instances of planar graphs, the GT Algorithm could also find near-optimal cylinder minors.

1.3 Organization of the thesis

The rest of the thesis is organized as follows. In Chapter 2, we give the definitions and notation that will be used in the thesis. Bidimensionality theory and bidimensionality theory based algorithms are reviewed in Chapter 3. We present the exact algorithm for the largest cylinder minor in planar graphs and its computational results in Chapter 4. The fast approximation algorithms for computing large grid minors are studied in Chapter 5. The algorithm for the planar longest path problem and its computational results are reviewed in Chapter 6. In the final chapter, we give the summary and the future work of the thesis.

Chapter 2

Preliminaries

2.1 Approximation algorithms and approximation ratio

For an NP-hard problem, there is no polynomial time exact algorithm to solve it unless P=NP. One way to tackle an NP-hard problem is to gain efficiency by sacrificing the accuracy of solutions. Much research has been done for developing efficient algorithms that provide near-optimal solutions for hard problems. We follow the book by Gonzalez [30] for the notation and definitions of approximation algorithm and approximation ratio.

For an algorithm A, given an instance I of problem P, we denote by $f_A(I)$ the objective function value of the solution returned by algorithm A for problem instance I, and by $f^*(I)$ the objective function value of an optimal solution for I.

An *approximation ratio*, denoted by ρ is used to describe the accuracy of algorithms. For every problem instance I of problem P, if an algorithm A satisfies $\frac{f_A(I)}{f^*(I)} \leq \rho$ for minimization problems, and $\frac{f^*(I)}{f_A(I)} \leq \rho$ for maximization problems, we call A a ρ-approximation algorithm, and refer to ρ as the *approximation ratio* of algorithm A.

For every instance I of problem P, if an algorithm A always returns the optimal solution, we call A an *exact algorithm* (i.e., $\rho = 1$). If ρ is greater than 1, A is called an *approximation algorithm*, and the closer to 1, the better the solution returned by the algorithm.

2.2 Notation and terminology

Our notation is mainly adopted from Gu and Tamaki [32]. In this thesis, graphs are undirected simple graphs unless otherwise stated. For a graph G, we denote the vertex set of G

by $V(G)$ and the edge set of G by $E(G)$. For each element $e \in E(G)$, we view e as a subset of $V(G)$ with two elements. We may use e or $V(e)$ to denote the set of vertices in edge e. For a subset $A \subseteq E(G)$ of edges we denote by $V(A) = \bigcup_{e \in A} V(e)$ the set of vertices in edges of A. We say that a vertex v and an edge e are incident to each other if $v \in V(e)$. We say that two edges e_1 and e_2 are incident to each other if $V(e_1) \cap V(e_2) \neq \emptyset$. A *vertex-cut* of G is a set of vertices such that the removal of these vertices disconnects G into at least two connected components.

A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. To *contract* an edge e in G is to remove e from G, identify the two end vertices of e with a single new vertex, and make all edges incident to e be incident to this new vertex. An example of edge contraction is shown in Figure 2.1. Notice that edge contraction may result in parallel edges between two vertices. In the applications discussed in this thesis (i.e., cylinder/grid minor, branch-decomposition), parallel edges do not affect the results. Therefore, when we discuss edge contractions in the rest of this thesis, we simply replace parallel edges between two vertices with one edge. A graph H is a *minor* of G if H can be obtained from a subgraph of G through a sequence (maybe empty) of edge contractions.

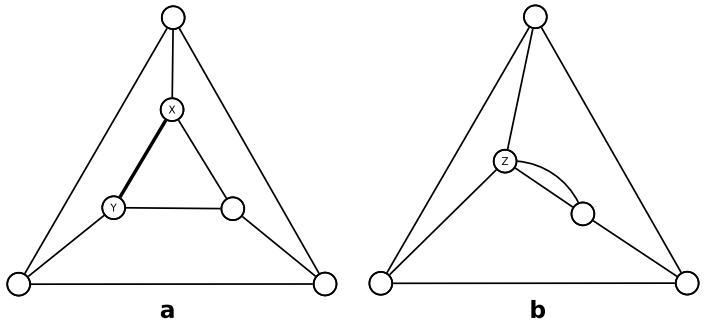

Figure 2.1: An example of edge contraction. The multigraph (a graph with parallel edges) in (b) is obtained from the graph in (a) by contracting edge $\{x, y\}$.

For a subset $A \subseteq E(G)$ of edges, the subgraph $G[A]$ of G induced by A is the graph with edge set A and vertex set $V(A)$. For a subset $U \subseteq V(G)$ of vertices, the subgraph $G[U]$ induced by U is the graph with vertex set U and edge set $\{\{u, v\} | u, v \in U, \{u, v\} \in E(G)\}$. For a subset $A \subseteq E(G)$ of edges, we denote $E(G) \setminus A$ by $\overline{A}$. A *separation* of graph G is a pair $(A, \overline{A})$ of subsets of $E(G)$. For each $A \subseteq E(G)$, we denote by $\partial(A)$ the vertex set

$V(A) \cap V(\overline{A})$. The *order* of a separation $(A, \overline{A})$ is $|\partial(A)| = |\partial(\overline{A})|$. Figure 2.2 (a) shows a graph G and Figure 2.2 (b) shows a separation $(A, \overline{A})$ of G, where $A = \{e_0, e_1, e_4\}$ and $\overline{A} = \{e_2, e_3, e_5, e_6, e_7\}$, $G_1 = G[A]$ and $G_2 = G[\overline{A}]$, $\partial(A) = \{1, 2, 4\}$ and the order of the separation is 3.

A *branch-decomposition* of a graph G is a pair (ϕ, T) where T is a tree in which each internal vertex has degree 3 and ϕ is a bijection from the set of leaves of T to $E(G)$. Consider a link e of T and let L_1 and L_2 denote the sets of leaves of T in the two respective subtrees of T obtained by removing e. We say that the separation $(\phi(L_1), \phi(L_2))$ is induced by the link e of T. We define the *middle set* of the separation induced by link e, denoted by $\mathrm{mid}(e)$, as $\partial(\phi(L_1))$ (i.e., $\mathrm{mid}(e) = \partial(\phi(L_1)) = \partial(\phi(L_2))$). The *width* of a branch-decomposition (ϕ, T) is the largest order of the separations induced by links of T. The *branchwidth* $\mathrm{bw}(G)$ of G is the minimum width of all branch-decompositions of G. In the rest of this thesis, we identify a branch-decomposition (ϕ, T) with the tree T, leaving the bijection implicit and regarding each leaf of T as an edge of G. Figure 2.3 shows a branch-decomposition of G in Figure 2.2 (a). The separation induced by the link f is the separation shown in Figure 2.2 (b).

A graph is *planar* if it can be drawn on a sphere without crossing edges. More precisely, we define the planar embedding of a graph in the following manner. Let Σ be a sphere. A set S of points in Σ is a *topological segment* of Σ if it is homeomorphic to an open segment $\{(x, 0) | 0 < x < 1\}$ on the sphere. For a topological segment S, we denote by $\overline{S}$ the closure of S and by $\mathrm{bd}(S) = \overline{S} \setminus S$ the two end points of S. A *planar embedding* of a graph G is a mapping γ from each vertex $v \in V(G)$ to a point p of Σ and from each edge $e \in E(G)$ to a topological segment S of Σ satisfying the following properties.

1. For $v \in V(G)$, $\gamma(v)$ is a point of Σ and, for distinct $u, v \in V(G)$, $\gamma(u) \neq \gamma(v)$.
2. For each edge $e = \{u, v\} \in E(G)$, $\gamma(e)$ is a topological segment with two end points $\gamma(u)$ and $\gamma(v)$.
3. For distinct $e_1, e_2 \in E(G)$, $\overline{\gamma(e_1)} \cap \overline{\gamma(e_2)} = \{\gamma(v) | v \in V(e_1) \cap V(e_2)\}$.

A graph G is *planar* if it has a planar embedding γ. For a planar graph G and a planar embedding γ of G, we call $(\gamma(V(G)), \gamma(E(G)))$ a *plane graph* , denoted by (G, γ), where $\gamma(V(G)) = \{\gamma(v) | \forall v \in V(G)\}$ and $\gamma(E(G)) = \{\gamma(e) | \forall e \in E(G)\}$. We also denote the plane graph (G, γ) by G, leaving the embedding γ implicit.

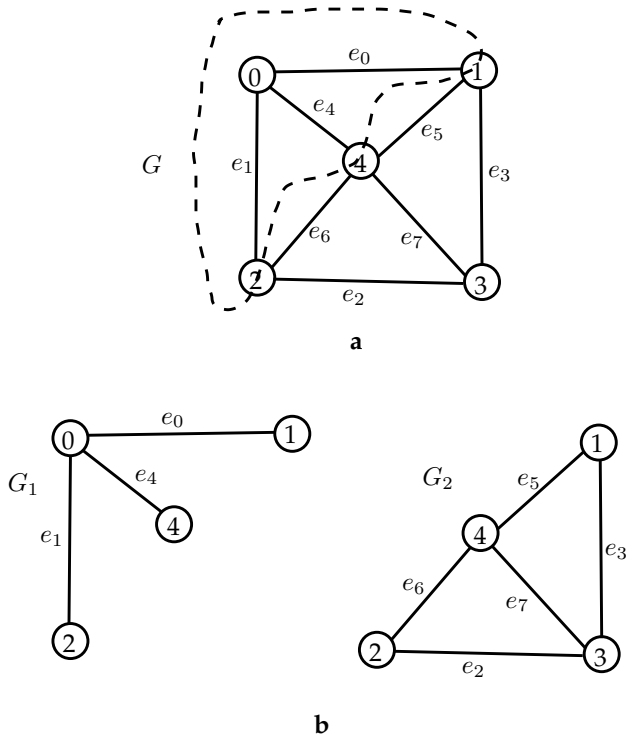

Figure 2.2: A separation $(A, \overline{A})$ of a graph G, where $A = \{e_0, e_1, e_4\}$ and $\overline{A} = \{e_2, e_3, e_5, e_6, e_7\}$. $G_1 = G[A]$, and $G_2 = G[\overline{A}]$.

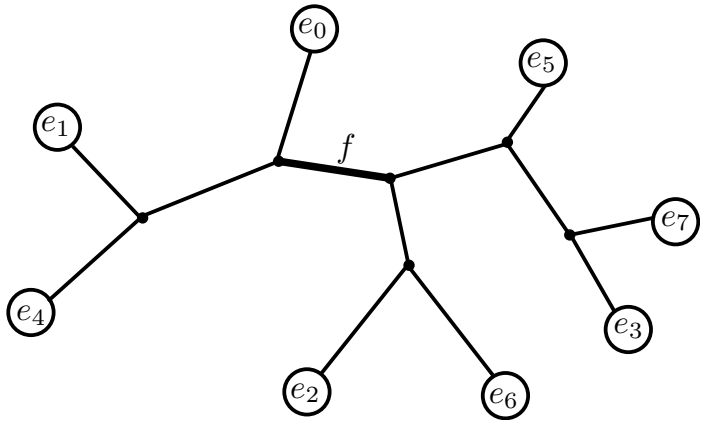

Figure 2.3: A branch-decomposition of G in Figure 2.2 (a).

We call each connected component of $\Sigma \setminus \bigcup_{x \in V(G) \cup E(G)} \gamma(x)$ a *face* of G. We denote by $F(G)$ the set of faces of G. For each $v \in V(G)$ ($e \in E(G)$, respectively), we call $\gamma(v)$ ($\gamma(e)$, respectively) a vertex (an edge, respectively) of plane graph G. We do not distinguish a vertex v (an edge e, respectively) from its embedding $\gamma(v)$ ($\gamma(e)$, respectively) if the context is clear.

Let G be a plane graph on Σ. We say a curve on the sphere Σ is *normal* if it does not intersect with itself nor any edge of G. A *G-noose* is a closed normal curve on Σ. A G-noose ν separates Σ into two regions R_1 and R_2, and induces a separation $(A, \overline{A})$ of G, where $A = \{e \in E(G) | \gamma(e) \subseteq R_1\}$, and $\overline{A} = \{e \in E(G) | \gamma(e) \subseteq R_2\}$. A separation of G is *noose-induced* if it is induced by some G-noose. A branch-decomposition T of G is a *sphere-cut decomposition* (*sc-decomposition* for short) if every separation induced by a link of T is noose-induced [25]. Every plane graph G has an sc-decomposition of width $\mathrm{bw}(G)$, and such a decomposition can be found in $O(n^3)$ time [33, 48].

For a normal curve ν of G, we add a unique point on ν for each face it intersects and there are a point for each vertex it intersects. These points for faces and points for vertices divide ν into several segments. We define the *length* of a normal curve ν as half of the number of segments in ν. For example, in Figure 2.2 (a), the dashed line represents a G-noose of length 3. A *noose-induced separation with order l* of G indicates that it is induced by a G-noose of length l.

For vertices or faces $x, y \in V(G) \cup F(G)$, the *normal distance* between x and y, denoted by $\mathrm{dist}(x, y)$, is the length of a normal curve on Σ of the smallest length between a point of x and a point of y. Notice that for $x \in V(G)$ and $y \in F(G)$, $\mathrm{dist}(x, y)$ is a $p/2$ for some odd integer p, but for $x, y \in V(G)$ or $x, y \in F(G)$, $\mathrm{dist}(x, y)$ is an integer. The length of a G-noose is also an integer. For a set of vertices or faces $X, Y \subseteq V(G) \cup F(G)$, $\mathrm{dist}(X, Y) = \min_{x \in X, y \in Y} \mathrm{dist}(x, y)$. We also use $\mathrm{dist}(x, Y)$ to denote $\mathrm{dist}(\{x\}, Y)$, and $\mathrm{dist}(X, y)$ to denote $\mathrm{dist}(X, \{y\})$.

Let ν be an oriented G-noose. We denote by $R(\nu)$ one of the two connected components of $\Sigma \setminus \nu$ which is on the right of ν when we proceed along it according to its orientation. If we remove the vertices and edges in $R(\nu)$, and add the segments of ν between the vertices on ν as new edges, then we get a new plane graph with a new face $f_0 = R(\nu)$. We denote this new plane graph by $G|\nu$. An example of getting $G|\nu$ from G and ν is shown in Figure 2.4.

A *path* P of a graph G is a sequence $v_0, e_1, v_1, e_2, ..., e_k, v_k$, where $v_i \in V(G)$, $e_i \in E(G)$,

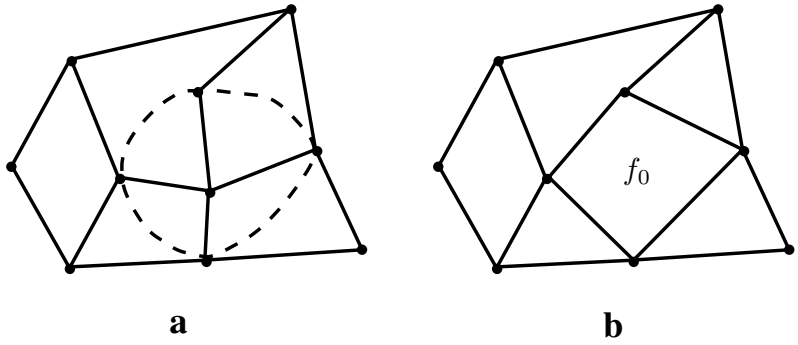

Figure 2.4: An example of getting $G|\nu$ from G and ν. G is shown in (a) and ν is shown in dashed curve with clockwise orientation. $G|\nu$ with new face f_0 is shown in (b).

$v_0 \neq v_k$, and each vertex/edge appears at most once in P. If $v_0 = v_k$ in this sequence, we call the sequence a *cycle*. The *length* of a path/cycle is the number of edges in it.

Figure 1.1 gives an intuitive view of grid and cylinder, the precise definitions of which are given as follows. For integers $g \geq 3$ and $h \geq 1$, we define a $(g \times h)$-grid, denoted by $G_{g,h}$, as $V(G_{g,h}) = \{(i,j)|0 \leq i < g, 0 \leq j < h\}$, $\{(i,j),(i',j')\} \in E(G_{g,h})$ if $|i-i'|+|j-j'| = 1$. We also define a $(g \times h)$-cylinder, denoted by $C_{g,h}$, as $V(C_{g,h}) = \{(i,j)|0 \leq i < g, 0 \leq j < h\}$, $\{(i,j),(i',j')\} \in E(C_{g,h})$ if $(i-i') \mod g \equiv 1$ and $j = j'$ or $i = i'$ and $|j-j'| = 1$. In $C_{g,h}$, for each $0 \leq j < h$, we call the cycle formed by all the vertices in $\{(i,j)|0 \leq i < g\}$ a *row cycle* of $C_{g,h}$. For each $0 \leq i < g$, we call the path formed by all the vertices in $\{(i,j)|0 \leq j < h\}$ a *column path* of $C_{g,h}$. If a graph G contains $G_{g,h}$ ($C_{g,h}$, respectively) as a minor, we say that $G_{g,h}$ ($C_{g,h}$, respectively) is a *grid minor* (*cylinder minor*, respectively) of G. The size of the largest grid minor of G, denoted by $\mathrm{gm}(G)$, is the largest integer g such that G contains a $(g \times g)$-grid as a minor. Let $\mathrm{cm}(G)$ be the largest g such that G contains $C_{g,\lceil g/2 \rceil}$ as a minor.

For a path $v_0, e_1, v_1, e_2, ..., e_k, v_k$ where $v_0 \neq v_k$, we call v_0 and v_k the *end vertices* and all other vertices the *internal vertices* of the path. Two paths in a graph are *internal vertex-disjoint* if they do not share a common internal vertex, while they are called *vertex-disjoint* if they have no vertex in common. A set of paths is vertex-disjoint (internal vertex-disjoint, respectively) if every pair of paths in the set is vertex-disjoint (internal vertex-disjoint, respectively). Two cycles are vertex-disjoint if they do not share a common vertex. A set

of cycles is vertex-disjoint if every pair of cycles in the set is vertex-disjoint. Similarly, two cycles are *edge-disjoint* if they do not share a common edge. A set of cycles is edge-disjoint if every pair of cycles in the set is edge-disjoint.

The *longest path problem* is, given a graph G, to find a path in G of the longest length. The decision version of the longest path problem, also known as the *k-longest path problem*, is to determine, given a graph G and an integer k, whether there is path in G of length $\geq k$. We also refer to the longest path problem in planar graphs as the *planar longest path problem*.

Chapter 3

Bidimensionality Theory Based Algorithms

3.1 Parameterized problems and bidimensionality theory

A parameter χ of a graph G is a function mapping G to a non-negative integer. The *parameterized problem* associated with χ is to determine for some fixed integer k, whether $\chi(G) = k$ [24]. For example, the parameterized longest path problem is to determine, for a fixed integer k, whether G has a path of length k. Notice that the parameterized longest path problem and the decision version of the longest path problem can be easily reduced to each other.

For some problems, the running time of an algorithm depends mainly on the value of a parameter rather than on the input instance size. Therefore, if the parameter value is small, an algorithm may be practical even if the input instance size is large. Formally, if a parameterized problem can be solved in time $O(f(k) \cdot n^{O(1)})$, where n is the input size of the problem and f is a computable function of k independent of n, we say that this problem is *fixed-parameterized tractable* [26]. If $f(k)$ is subexponential in k (e.g., $f(k) = 2^{O(\sqrt{k})}$), we call the algorithm of running time $O(f(k) \cdot n^{O(1)})$ for solving the parameterized problem a *subexponential parameterized algorithm*. Some NP-complete problems, such as planar k-vertex cover problem, planar k-dominating set problem, planar k-longest path problem, etc., can be solved by subexponential parameterized algorithms [22].

Bidimensionality theory is introduced as a tool to develop these subexponential parameterized algorithms [18, 24]. The intuition behind bidimensionality theory is that some parameters do not increase under taking minors or contractions in a graph. Moreover, the value of the parameter depends on the "area" $g \times h$ of a grid $G_{g,h}$. We can obtain a lower bound on the parameter by finding a large grid minor of an input graph.

Another important ingredient of bidimensionality theory is the linear upper bound on branchwidth in terms of the size of the largest grid minor for some classes of graphs including planar graphs. If we cannot find a large grid minor for a graph G, it implies that G has a small branchwidth, and the problem could be solved by branch-decomposition based dynamic programming algorithms typically in $O(2^{O(\mathrm{bw}(G))}n^{O(1)})$ time. Since the size of the largest grid minor is on the order of the square root of the size of the graph (i.e., $\mathrm{gm}(G) \leq \sqrt{|V(G)|}$), according to the relationship between $\mathrm{bw}(G)$ and $\mathrm{gm}(G)$, it follows that the branch-decomposition based dynamic programming algorithms are subexponential in k for problems with parameter value $k = \Theta((\mathrm{gm}(G))^2)$.

Below we adopt the formal definitions of bidimensionality theory introduced by Dorn et al. [24].

Definition 3.1.1. *A parameter χ is minor bidimensional with density δ if*

1. *χ does not increase under taking minors, and*
2. *for a $(g \times g)$-grid R, $\chi(R) = (\delta g)^2 + o((\delta g)^2)$.*

Definition 3.1.2. *A parameter χ is contraction bidimensional with density δ if*

1. *χ does not increase under contractions*
2. *for a $(g \times g)$-grid R, $\chi(R) = (\delta_R \cdot g)^2 + o((\delta_R \cdot g)^2)$.*
3. *δ is the smallest δ_R among all partially triangulated $(g \times g)$-grids.*

A parameter is *bidimensional* if it is either minor bidimensional or contraction bidimensional. For example, the number of vertices and the number of edges of a graph are bidimensional parameters, although this is not a very interesting example.

Consider the vertex cover problem for example. It is obvious that the size of a vertex cover does not increase under taking minors. The size of vertex cover on a $(g \times g)$-grid is at least $g^2/2$. Thus we conclude that the parameter *vertex cover* is bidimensional with density $\delta = 1/\sqrt{2}$.

The size of a dominating set does not hold the non-increasing property under taking minors. However, the parameter does not increase under contractions. We use the partially triangulated $(g \times g)$-grid (a planar graph obtained from the $(g \times g)$-grid by adding some edges, an example is shown in Figure 3.1) to analyze the bidimensionality of the parameter *dominating set*. In every partially triangulated $(g \times g)$-grid R, an inner vertex has a closed neighborhood of at most 9 vertices. It follows that the size of a dominating set in R is at least $\frac{(g-2)^2}{9}$. Therefore the parameter *dominating set* is bidimensional with density $\delta = 1/3$.

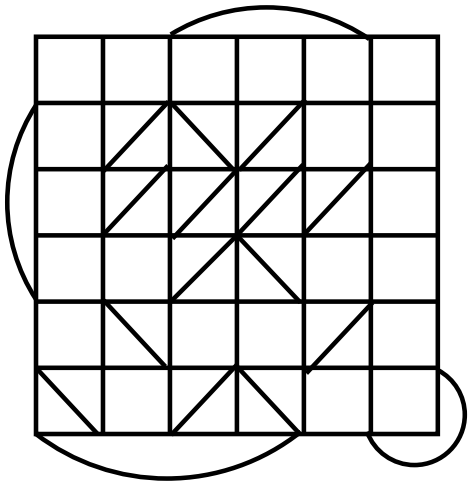

Figure 3.1: An example of partially triangulated (6×6)-grid.

A path of the longest length in a $(g \times g)$-grid is obtained by traversing all vertices, as shown in Figure 3.2, and has length $g^2 - 1$. Since the length of the longest path does not increase under taking minors, the parameter *longest path* is bidimensional with density $\delta = 1$.

Other examples of bidimensional parameters are *feedback vertex set* which has density $\delta \in [1/2, 1/\sqrt{2}]$, and *minimum maximal matching* which has density $\delta \in [1/\sqrt{8}, 1/\sqrt{2}]$ [24].

The relationship between branchwidth and the size of the largest grid minor is important in bidimensionality theory. We focus on planar graphs in this discussion. It is well known that $\mathrm{gm}(G) \leq \mathrm{bw}(G) \leq 4\mathrm{gm}(G)$ [44]. Recently, Gu and Tamaki [32] have improved this result to $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$, which implies the following theorem.

Theorem 3.1.1. *[32] Let $l \geq 1$ be an integer. Every planar graph of branchwidth $\geq l$*

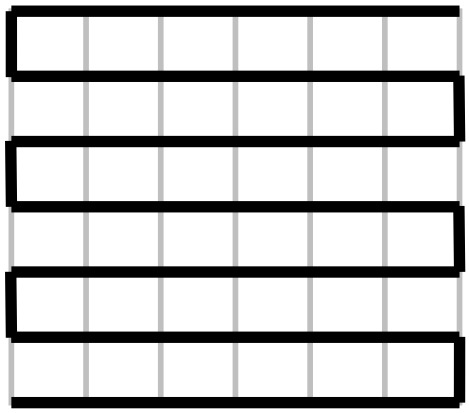

Figure 3.2: A longest path, shown in bold lines, on a (6×6)-grid.

contains a $(\lceil (l-1)/3 \rceil \times \lceil (l-1)/3 \rceil)$*-grid as a minor.*

Gu and Tamaki's proof for Theorem 3.1.1 is constructive, and implies an algorithm (the GT Algorithm) which, given an integer g and a planar graph G, either finds a $(g \times g)$-grid as a minor or asserts that $\mathrm{bw}(G) \leq 3g$. We present the details of this algorithm in Chapter 5.

The definition of bidimensionality implies $\chi(G) \geq (\delta \cdot \mathrm{gm}(G))^2$, which follows $\mathrm{gm}(G) \leq \frac{1}{\delta}\sqrt{\chi(G)}$. Together with Theorem 3.1.1, branchwidth can be bounded sublinearly to bidimensional parameters:

$$\mathrm{bw}(G) \leq \frac{3}{\delta} \cdot \sqrt{\chi(G)}.$$

Using branch-decomposition based techniques, some parameters can be computed in $O(2^{\alpha \cdot \mathrm{bw}(G)} n^{O(1)})$ time, where n is the number of vertices of G and α is a constant. Let G be a planar graph, and χ be a bidimensional parameter that can be computed in $O(2^{\alpha \cdot \mathrm{bw}(G)} n^{O(1)})$ time. Dorn et al. [24] give a general framework of bidimensionality theory based algorithms for solving the parameterized problems in G associated with a parameter χ of value k in $O(n^3 + 2^{O(\sqrt{k})} n^{O(1)})$ time as follows.

An optimal branch-decomposition and the branchwidth $\mathrm{bw}(G)$ of a planar graph G can be computed in $O(n^3)$ time. If $\mathrm{bw}(G) \geq \frac{3}{\delta} \cdot \sqrt{k}$, then the answer to the associated parameterized problem with parameter value k is "NO" for a minimization problem and "YES" for a maximization problem. Otherwise, $\chi(G)$ can be computed in $O(2^{\alpha \cdot \mathrm{bw}(G)} n^{O(1)}) =$

$O(2^{\alpha \cdot \frac{3}{\delta}\sqrt{k}} n^{O(1)})$ time.

3.2 A case study: An algorithm for the longest path problem

As discussed above, the length of the longest path is a bidimensional parameter. Given a planar graph G and an integer k, the longest path problem is to determine whether there exists a path of length $\geq k$. To solve this problem using bidimensionality theory, we first compute an optimal branch-decomposition of G. If $\text{bw}(G) \geq 2\sqrt{2(k+1)}$, we answer "YES". Otherwise, we use a branch-decomposition based dynamic programming algorithm to compute the longest path in G.

We first review a branch-decomposition based algorithm for the longest path problem in general graphs. The idea of the algorithm is similar to that described by Cook and Seymour [8] for the TSP problem. A more efficient algorithm for the longest path problem in a planar graph G can be obtained using some geometric properties of G embedded on a sphere. This algorithm will be discussed in Chapter 6.

Let T be a branch-decomposition of a graph G. We first convert T to a rooted binary tree, by replacing an arbitrary link $e = \{u, v\}$ of T with three links $\{u, s\}$, $\{s, v\}$ and $\{s, r\}$. The new vertex r is called the *root* of T. Let $\text{mid}(\{s, r\}) = \emptyset$. An example of the rooting process is shown in Figure 3.3, where the original branch-decomposition tree is shown in Figure 2.3, and the link between the leaf e_0 and the internal vertex u is replaced by three links. Removing a link e in T leaves two subtrees of T. We call the subtree containing the root r the *upper part* and the other the *lower part*. We denote by G_e the subgraph of G induced by the edges associated with the leaves of the lower part of e.

Let $(A, \overline{A})$ be a separation of G. A partial solution Q of the longest path problem in $G[A]$ is a set of vertex-disjoint paths in $G[A]$ satisfying the following properties.

1. Every path in Q has at least one end vertex in $\partial(A)$.
2. At most two paths in Q could have only one end vertex in $\partial(A)$. Other paths have both end vertices in $\partial(A)$.

We define the *length* of a partial solution Q as the sum of the lengths of all paths in Q.

Let $v_1, v_2, ..., v_l$ be the vertices of $\partial(A)$. Each vertex v_i $(1 \leq i \leq l)$ must fall into one of the following three cases: (1) v_i does not belong to any path in Q; (2) v_i is an end vertex of some path in Q; (3) v_i is an internal vertex of some path of Q. In case (2), v_i is paired

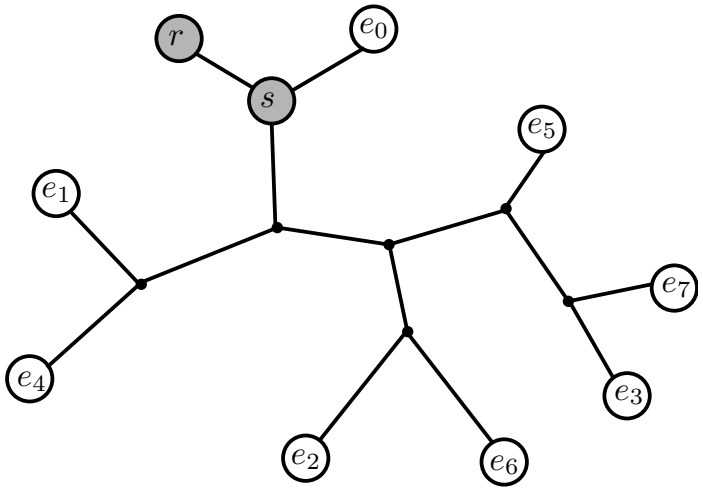

Figure 3.3: Converting the branch-decomposition of Figure 2.3 to a rooted binary tree.

to v_j, the other end vertex of the path containing v_i. If the path containing v_i has only one end vertex in $\partial(A)$, we set v_j to nil. We call a matching $\{(v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), ..., (v_{i_m}, v_{j_m})\}$ a *character*, denoted by d. Let $D = \{d_1, d_2, ..., d_k\}$ be the list of all possible characters of the vertices in $\partial(A)$. For each character $d \in D$, there is an associated variable $C(d)$ that stores the longest length of all possible partial solutions that can be represented by d. We denote by $C(D)$ the list of length variables $C(d)$ for all $d \in D$.

The dynamic program proceeds from leaves upwards to the root in T. For a leaf vertex e, the corresponding separation contains only one edge $e = \{u, v\}$, which is either included in a partial solution or not. Thus there are only two valid characters for a leaf vertex, one of which is $\{(u, v)\}$ and the other is empty (containing no pairs), and D and $C(D)$ can be easily computed for leaves. We mark each leaf vertex as "merged". For every internal vertex v of T, we denote by p_v the parent (the first vertex on the path from v to r) of v, and by l_v and r_v the two children (the first vertex on the path from v to the leaves) of v. We call l_v the left child of v and r_v the right child. For an internal vertex v of T, we merge it if both its children have been merged.

When we process an internal vertex v of T, let D_l be the list of characters of its left child, and D_r of its right child. For each character $d_1 \in D_l$ and $d_2 \in D_r$, when the partial solutions associated with d_1 and d_2 can be merged into a character d_p to form a partial solution in $G_{\{p_v, v\}}$, we add d_p into the character list of v if d_p has not been added it yet,

otherwise, if $C(d_1) + C(d_2) > C(d_p)$, we update $C(d_p)$. After processing each pair of a character from D_l and a character from D_r, we have the character list D_p and list of the length variable $C(D_p)$ of v, and mark v as merged. Finally, when we reach the root r of T, the partial solutions become the paths of G. The longest of these is the solution of the longest path problem.

The running time of the dynamic programming algorithm is quadratic in the largest number of possible characters of a separation. For a separation, we define a *candidate set* as a set of vertices in the middle set that are involved in some particular character (i.e., it appears as an end vertex of some path in some particular partial solution). Many characters can be derived from one candidate set. For a separation of order l, the number of candidate sets is 2^l. For each candidate set, each character can be viewed as a partition of it. The number of possible characters of a candidate set of m vertices is the number of partitions of m elements and is bounded by the m-th Bell number [2, 46]:

$$B_m < \left(\frac{0.792m}{\ln(m+1)}\right)^m \approx c^{m \log m},$$

where c is some constant. Since $m \leq l$, the number of possible characters of a separation of order l is bounded by $O(2^l c^{l \log l})$. If we can compute a branch-decomposition with width l of G in $f(n)$ time, where n is the number of vertices of G, then the total running time for solving the longest path problem in G is $O(2^{O(l \log l)} n^{O(1)} + f(n))$. In Chapter 6, we introduce the non-crossing property of the sc-decomposition of planar graphs. Using this property, the log factor in the exponent can be eliminated.

Chapter 4

Exact Algorithm for the Largest Cylinder Minor

Theorem 3.1.1 gives a lower bound on the size of the largest grid minor in terms of branch-width. In theory, a better lower bound implies faster subexponential parameterized algorithms, while in practice, computing large grid minors is useful for solving the parameterized problems for some parameter value k. Therefore, we are interested in finding large grid minors.

The decision version of the problem of finding the largest grid minor in general graphs is known to be NP-complete [31]. Whether it remains NP-complete for planar graphs or not is still not known. The algorithm implied by the proof of Theorem 3.1.1 is a 3-approximation algorithm for finding the largest grid minor in planar graphs, which will be discussed in Chapter 5. In this chapter, we develop an exact algorithm for computing the largest cylinder minor in planar graphs. The running time of this algorithm is not polynomial.

Let $C_{g,h}$ be a cylinder embedded on a sphere Σ. We call the cycle with the vertex set $\{(i, h-1)|0 \leq i < g\}$ ($\{(i,0)|0 \leq i < g\}$) the *top cycle* (*bottom cycle*, respectively). We call the face incident only to the vertices of the top cycle (bottom cycle) the *top face* (*bottom face*, respectively). The set of vertices in each row cycle of $C_{g,h}$ can be considered as a vertex-cut separating the top face from the bottom face. We say that each row cycle of $C_{g,h}$ induces such a vertex-cut. These vertex-cuts induced by the row cycles are pairwise vertex-disjoint. The intuition of the exact algorithm is as follows. Assume that a planar graph has $C_{g,h}$ as a minor. Then G has a vertex-cut of order at least g induced by the top

cycle of $C_{g,h}$. We first find the vertex-cut induced by this top cycle, then we identify the vertex-cuts induced by the rest of row cycles of $C_{g,h}$ by a technique of computing "contours". A contour is a set of edge-disjoint cycles of G. Finally we find $C_{g,h}$ from the h vertex-cuts. We enumerate all of the vertex-cuts in G of a certain order to find the one induced by the top cycle of a largest cylinder minor of G. An auxiliary graph is built to reduce the vertex-cut enumeration problem to the cycle enumeration problem. To complete the algorithm, we need techniques for enumerating cycles of a certain length in a graph and techniques for computing the contours. The former will be presented in Section 4.1 and the latter in Section 4.2.

4.1 Enumerating cycles

Let G be a plane graph. In order to represent a vertex-cut of G as a cycle, we construct an auxiliary graph G_A as follows.

1. $V(G_A) = V(G)$. $E(G_A) = E(G) \cup E'$, where $E' = \{\{u, v\} | u, v \in V(G)$, u and v are not adjacent, u and v are incident to a same face $f \in F(G)\}$.

2. For an edge $\{u, v\} \in E'$, where u and v are incident to the same face $f \in F(G)$, we say that $\{u, v\}$ is associated with f.

Notice that G_A is not necessarily planar. We say that a cycle $v_0, e_1, v_1, e_2, ..., e_m, v_m$ in G_A is *cutable*, if no two edges in this cycle are associated with the same face. Every cutable cycle $v_0, e_1, v_1, e_2, ..., e_m, v_m$ in G_A can be identified by a vertex-cut $\{v_0, v_1, ..., v_m\}$ in G. We have the following observation.

Observation 4.1.1. *A cutable cycle $v_0, e_1, v_1, e_2..., e_m, v_m$ in G_A can be identified by a G-noose ν in G defined as follows: the segment of ν between v_i and v_{i+1} is e_{i+1} if $e_{i+1} \in E'$, otherwise is a normal curve between v_i and v_{i+1} in one of the two faces incident to e_{i+1} in G.*

Therefore, enumerating the vertex-cuts of order k in G can be reduced to enumerating the cycles of length k in G_A.

Our method for enumerating cycles is based on Byers and Waterman [7]. We use a stack to keep track of sub-paths in the procedure, and use P to record the path for each entry.

Each entry of the stack has three attributes: (1) x, a next-to-last vertex; (2) y, a last-vertex; (3) c, the length of the path $(1, ..., x, y)$ having a sub-path $(1, ..., x)$ in P.

The procedure for enumerating all cycles of G_A containing vertex v of length k is shown in Algorithm 4.1.1.

Algorithm 4.1.1 Enumerate all cycles of graph G_A containing vertex v of length k

```
1: procedure ENUMERATE-CYCLE(G_A, k, v)
2:     create a stack S
3:     for all vertices u ≠ v in G_A do
4:         set f(u) to the length of the shortest path from v to u in G_A
5:     end for
6:     P ← (v)
7:     x ← v
8:     for every edge {v, y} that satisfies 1 + f(y) ≤ k do
9:         c ← 1
10:        push entry (x, y, c) onto S
11:    end for
12:    while S is not empty do
13:        (x, y, c) ← POP(S)
14:        replace P = (v, ..., x) by P = (v, ..., x, y)
15:        if y == v then
16:            output P and c
17:        else
18:            x ← y
19:            d ← c
20:            for each edge (x, y) such that d + 1 + f(y) ≤ k do
21:                c ← d + 1
22:                push entry (x, y, c) onto S
23:            end for
24:        end if
25:    end while
26: end procedure
```

4.2 Cylinder minor testing

A cylinder consists of row cycles and column paths, both of which are vertex-disjoint. We define contours to identify the row cycles in a cylinder. Let $F \subseteq F(G)$ be a set of faces. A *contour* of height i from F, denoted by $\text{cont}(F, i)$, is the subgraph of G induced by the set of edges $e \in E(G)$ such that e is incident to both a face r with normal distance $\text{dist}(r, F) = i$

and a face r' with normal distance $\text{dist}(r', F) = i+1$. For a face $f \in F(G)$, we use $\text{cont}(f, i)$ to denote $\text{cont}(\{f\}, i)$. An example of a contour is shown in Figure 4.1. The following proposition for the property of contours will be used later.

Proposition 4.2.1. *[32] Let G be a plane graph, $F \subseteq F(G)$ a set of faces of G, and $i \geq 0$ an integer. Then,* $\text{cont}(F, i)$ *consists of edge-disjoint cycles.*

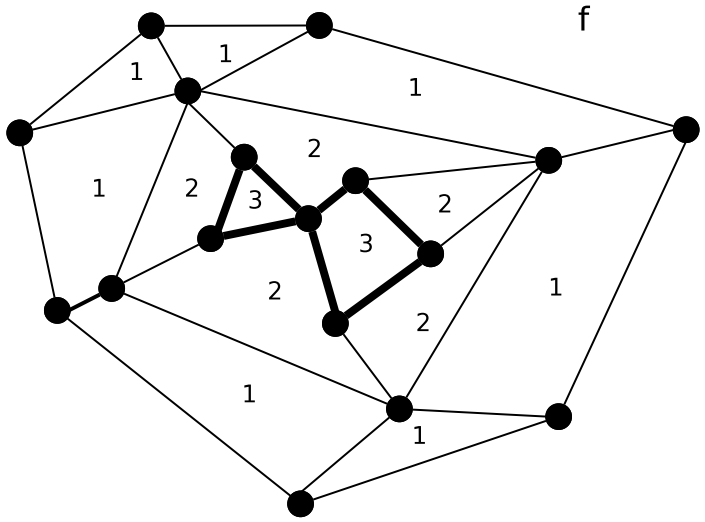

Figure 4.1: An example of the contour $\text{cont}(f, 2)$, the edges of which are shown in bold lines. The face f is the outer face, and the normal distance to f is labeled for each face.

For a face f, a contour $\text{cont}(f, i)$ consists of edge-disjoint cycles. The cycles in $\text{cont}(f, i)$ and those in $\text{cont}(f, j)$ are vertex-disjoint for $i \neq j$. For integers $g \geq 3$ and $h \geq 1$, let C be a cycle in $\text{cont}(f, h-1)$. Then C separates Σ into two regions, one of which does not contain f (denoted by R). Assume that there are g vertex-disjoint paths between f (the vertices incident to f) and R (the vertices incident to R). Then for every $0 \leq i < h$, these g paths intersect a cycle from $\text{cont}(f, i)$. The graph consisting of these g paths and h cycles has a $(g \times h)$-cylinder minor. We give the pseudocode for the cylinder minor finding algorithm in more precisely Algorithm 4.2.1.

Algorithm 4.2.1 takes a plane graph G, integers $g \geq 3$ and $h \geq 1$ as inputs. It returns "YES" if G contains a $(g \times h)$-cylinder as a minor, and "NO" otherwise. Line 19 in Algorithm 4.2.1 computes the number of vertex-disjoint paths between two faces. The problem of

Algorithm 4.2.1 Decide if G contains a $(g \times h)$-cylinder as a minor

```
1: procedure TEST-MINOR(G, g, h)
2:     construct the auxiliary graph G_A of G
3:     for each pair of faces f_s and f_t in G do
4:         compute the shortest path P from f_s to f_t
5:         for cg ← g to √|V(G)| do
6:             for each vertex v ∈ P do
7:                 call ENUMERATE-CYCLE(G_A, cg, v) to find the set C of all cycles in G_A
   containing v, keeping only the cycles c ∈ C which are cutable and cross P an odd
   number of times
8:                 for each cycle c ∈ C do
9:                     by Observation 4.1.1, get a noose ν_1 in G corresponding to c
10:                    orient ν_1 such that f_s ∈ R(ν_1)
11:                    get G|ν_1, and this results in a face f_0
12:                    compute the contour cont(f_0, h) (if it exists)
13:                    according to Proposition 4.2.2, cont(f_0, h) consists of edge disjoint cy-
   cles {c_1, c_2, ..., c_m} (notice that all these cycles are cutable)
14:                    for c_i ∈ {c_1, c_2, ..., c_m} do
15:                        by Obeservation 4.1.1, get a noose ν_2 corresponding to c_i
16:                        orient ν_2 such that f_t ∈ R(ν_2)
17:                        get G|ν_1|ν_2, and this results in a face f_1
18:                        compute the number of vertex disjoint paths from f_0 to f_1
19:                        if there are at least g vertex disjoint paths then
20:                            return "YES"
21:                        end if
22:                    end for
23:                end for
24:            end for
25:        end for
26:    end for
27:    return "No"
28: end procedure
```

finding a maximum cardinality set of vertex-disjoint paths in a planar graph can be reduced to the max-flow problem in linear time [34]. To prove the correctness of the algorithm, we need the following result.

Proposition 4.2.2. *[32] Let G be a plane graph and $g \geq 3$ and $h \geq 1$ be integers. Let f_1 and f_2 be two faces of G such that the following conditions are satisfied:*

1. *There is no separation (A, B) of G of order $< g$ such that $V(f_1) \subseteq V(A)$ and $V(f_2) \subseteq V(B)$.*
2. *$\text{dist}(f_1, f_2) \geq h$*

Then, G contains a $(g \times h)$-cylinder as a minor.

We have the following theorem.

Theorem 4.2.1. *Algorithm 4.2.1 outputs "YES" if and only if G contains a $(g \times h)$-cylinder as a minor.*

Proof. The "only if" part is straightforward. When the procedure outputs "YES", according to Proposition 4.2.2, G contains a $(g \times h)$-cylinder as a minor.

To prove the "if" direction, assume that G contains a $(g \times h)$-cylinder $C_{g,h}$ as a minor but the procedure outputs "NO". Let the bottom cycle of the cylinder be c_0. The length of c_0 is an integer between g and $\sqrt{|V(G)|}$. In Algorithm 4.2.1, line 5 and line 7 ensure that c_0 will be enumerated. Then we compute the contours $\text{cont}(r_0, h)$ of distance h form the face r_0 which is formed by c_0. The contour $\text{cont}(r_0, h)$ consists of edge-disjoint cycles $\{c_1, c_2, ..., c_m\}$. For some cycle c_i $(1 \leq i \leq m)$, the subgraph induced by the vertices and edges from c_0 to c_i is contained in $C_{g,h}$. Since the procedure does not terminate in line 20, there is a separation (A, B) of order $< g$ separating c_0 from c_i, which leads to a contradiction with the $(g \times h)$-cylinder. □

The time complexity depends on the number of cycles of a certain length containing a vertex in planar graphs (i.e., the for-loop in line 8 of Algorithm 4.2.1). There is no known polynomial bound on this number. Buchin et al. [6] show that the number of simple cycles (cycles that do not intersect themselves) in a planar graph is at most 2.8927^n, where n is the number of vertices in G. Therefore, the running time of Algorithm 4.2.1 is bounded exponentially.

4.3 Computational results

We implement Algorithm 4.1.1 and Algorithm 4.2.1 for computing the largest cylinder minor in planar graphs and test the algorithm on the graphs from TSPLIB [43]. The running time of this algorithm is not polynomial, and the implementation can handle only small instances of graphs.

The computer used for testing this algorithm has an Intel Pentium 4 3.00 GHz CPU and 2 GByte RAM. The operating system is Cent OS 5.5. The algorithm is implemented with C++.

Table 4.1: Computational results of the optimal largest cylinder minor algorithm.

graphs	# of edges	cm	time	time1	time2
eil51	140	7×4	1.21	0.04	0.03
eil76	215	8×5	0.07	0.04	0.03
pr76	218	8×6	4.4	0.05	0.03
rat99	279	7×6	0.78	0.11	553.61
rd100	286	$7 \times 6^*$	1.15	0.14	-
kroB100	284	$8 \times 6^*$	23.55	275.55	-
lin105	292	7×7	1.63	0.15	718.11
ch130	377	$9 \times 6^*$	271.22	0.14	-
pr144	393	$9 \times 4^*$	80.92	-	-

The results are tabulated in Table 4.1, where "cm" is the cylinder minor found by this optimal algorithm. The column "time" indicates the time (in seconds) for computing the cylinder minor. The columns "time1" and "time2" give the times for answering "NO" for computing $(g \times (h+1))$-cylinder minor and $((g+1) \times h)$-cylinder minor, respectively. For "time1" and "time2", the entries "-" mean that the algorithm cannot finish with 2 GByte memory. The entries of "cm" marked with "*" mean that the cylinder minor it computes is not known to be the largest, since either "time1" or "time2" is not available. The computational results show that the implementation of this exact algorithm can handle small planar graphs with less than 100 vertices with 2 GByte memory. In Chapter 5, we will use the results from Table 4.1 to evaluate the quality of cylinder minors found by the GT Algorithm.

Chapter 5

Approximation Algorithms for the Largest Grid Minor

5.1 The BGK Algorithm

The first linear relationship between branchwidth and the size of the largest grid minor is $\mathrm{gm}(G) \leq \mathrm{bw}(G) \leq 4\mathrm{gm}(G)$, which is given by Robertson et al. [44]. Based on the proof for $\mathrm{bw}(G) \leq 4\mathrm{gm}(G)$ by Robertson et al. [44], Dodlaender et al. [5] develop an efficient 4-approximation algorithm for computing the largest grid minor in planar graphs. We refer to this algorithm as *the BGK Algorithm* (after the authors Bodlaender, Grigoriev and Koster). The idea of their algorithm is as follows.

For a plane graph G, an arbitrary face $f \in F(G)$ incident to at least four edges is picked. If G is triangulated, an arbitrary edge is removed in order to form a face incident to at least four edges. The cycle consisting of edges incident to f is partitioned into four segments S_1, S_2, S_3 and S_4 of roughly equal size, where $V(S_1) \cap V(S_3) = \emptyset$ and $V(S_2) \cap V(S_4) = \emptyset$ (where $V(S_i)$ denotes the set of vertices in S_i). The algorithm tries to find g vertex-disjoint paths between $V(S_1)$ and $V(S_3)$ and g vertex-disjoint paths between $V(S_2)$ and $V(S_4)$. If these paths are found, the g vertex-disjoint paths between $V(S_1)$ and $V(S_3)$ and g vertex-disjoint paths between $V(S_2)$ and $V(S_4)$ form a $(g \times g)$-grid minor. Otherwise, a vertex-cut of order $< g$ is obtained separating $V(S_1)$ from $V(S_3)$ or separating $V(S_2)$ from $V(S_4)$. In this case, we enlarge f to this vertex-cut and repeat the above procedure until a $(g \times g)$-grid minor is found or the number of vertices in the remaining graph is less than g^2. In the latter case,

the branchwidth $\mathrm{bw}(G)$ of G is claimed to be at most $4g$.

Dodlaender et al. [5] test the performance of their algorithm on graphs from TSPLIB [43]. Their computational results show that their algorithm is practical. Gu and Tamaki improve the linear bound to $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ [32]. We study and implement the GT Algorithm to compare the quality of grid minors found by these two algorithms.

5.2 The GT Algorithm

For a face f in a plane graph G, let f' be a face of G with the maximum normal distance $\mathrm{dist}(f, f')$. It is known that a branch-decomposition of G with width at most $2 \times \mathrm{dist}(f, f')$ can be constructed in linear time [49]. Based on this result, Gu and Tamaki give an approximation algorithm for computing the branch-decomposition and grid minor of G.

Intuitively the algorithm works as follows. Let G be a plane graph and g, h be integers. We select a face f of G. If $\mathrm{dist}(f, f') < h$ then we compute a branch-decomposition of G with width at most $2h$. Otherwise, we find the contour $\mathrm{cont}(f, h-1)$ which is a set of edge-disjoint cycles. For simplicity, we assume that this contour has one cycle. The cycle partitions the sphere Σ into two regions R_1 and R_2 with f in one region, say f in R_1. We compute a minimum vertex-cut C of G which separates f from R_2. If $|C| \geq g$ then we find a $(g \times h)$-cylinder minor of G. Otherwise, let (A, B) be the separation of G induced by C (i.e., $C = \partial(A) = \partial(B)$) with $f \in F(G[A])$. We find a branch-decomposition T_A of $G[A]$ and apply the algorithm recursively to graph $G[B]$ taking the region which contains $G[A]$ as the face f of $G[B]$. The recursion finds either a $(g \times h)$-cylinder minor or a branch-decomposition T_B of $G[B]$. In the latter case, a branch-decomposition of G with width at most $\max\{k_A, k_B\} + |C|$ is constructed, where k_A is the width of T_A and k_B the width of T_B.

In the algorithm, it is often more convenient to work on a hypergraph consisting of $G[A]$ and a hyperedge e with $V(e) = C$ to represent $G[B]$ and a hypergraph consisting of $G[B]$ and a hyperedge e to represent $G[A]$.

A *hypergraph* G consists of a set $V(G)$ of vertices and a set $E(G)$ of *hyperedges* with each $e \in E(G)$ a subset of $V(G)$ of at least two elements. The *degree* of a hyperedge is the number of elements (vertices) it contains. If every hyperedge in a hypergraph has two elements, this hypergraph becomes a graph. The notation for planar embedding introduced in Chapter 2 can be adapted to hyperedges and hypergraphs if we use the notion of *topological disc*

$D = \{(x, y)|x^2 + y^2 < 1\}$ instead of topological segment and point. We denote the closure of D by $\overline{D}$ and the boundary of D by $\mathrm{bd}(D) = \overline{D} \setminus D$. Similarly, a planar embedding of a hypergraph G on a sphere Σ is a mapping γ from each element $x \in V(G) \cup E(G)$ to a topological disc D of Σ satisfying the following properties.

1. For $v \in V(G)$, $\gamma(v)$ is a topological disc of Σ, and for distinct $u, v \in V(G)$, $\overline{\gamma(u)} \cap \overline{\gamma(v)} = \emptyset$.

2. For each edge $e = \{u, v\} \in E(G)$, $\gamma(e)$ is a topological disc of Σ and for distinct $e_1, e_2 \in E(G)$, $\overline{\gamma(e_1)} \cap \overline{\gamma(e_2)} = \emptyset$.

3. For each $v \in V(G)$ and each $e \in E(G)$, $\gamma(v) \cap \gamma(e) = \emptyset$. Moreover, $\mathrm{bd}(\gamma(v)) \cap \mathrm{bd}(\gamma(e))$ is either an empty set if $v \notin e$ or a single segment of positive length otherwise.

Similarly to the definition of plane graph, we also define the *plane hypergraph*, denoted by G, leaving the embedding γ implicit. A *normal curve* is defined for plane hypergraph in the same way as in Section 2.2. Recall that when we define the length of a normal curve of a plane graph, we add a unique point for each face it intersects. For the normal curve of a plane hypergraph, we also add a unique point for each vertex it intersects, since vertices are represented as open discs rather than points. The definition of *normal distance* for plane hypergraph is the same as for plane graph. Figure 5.1 shows an example of a hypergraph, where solid areas represent hyperedges.

Let G be a hypergraph and (A, B) a separation of G. We denote by $G|A$ a hypergraph with $V(G|A) = V(B)$ and $E(G|A) = B \cup \{\partial(A)\}$. Intuitively, in $G|A$, we replace the sub-hypergraph of G induced by A with a single hyperedge $\partial(A)$.

For a plane hypergraph G and a hyperedge e_0 of G, we construct a plane hypergraph $G|e_0$ as follows. Let S_0 be the set of closures of connected components of $\mathrm{bd}(e_0) \setminus \bigcup_{v \in V(e_0)} v$. Each member of S_0 is an arc, with each endpoint in the boundary of a vertex incident to e_0. For each $s \in S_0$, let $\hat{s}$ be a band-shaped disc obtained by thickening s so as to keep a proper incidence as an edge with the vertices at the ends of s. Let $E_0 = \{\hat{s}|s \in S_0\}$ be a set of edges and let $V(G|e_0) = V(G)$ and $E(G|e_0) = E(G) \cup E_0 \setminus \{e_0\}$. We place edges in E_0 inside e_0 properly so that $F(G|e_0) = F(G) \cup \{r_0\}$, where r_0 is the closure of $e_0 \setminus \bigcup_{e \in E_0} e$. Intuitively, obtaining $G|e_0$ from G can be visualized as a procedure that removes the hyperedge e_0 and creates a face r_0 by "hollowing" e_0 (see Figure 5.2).

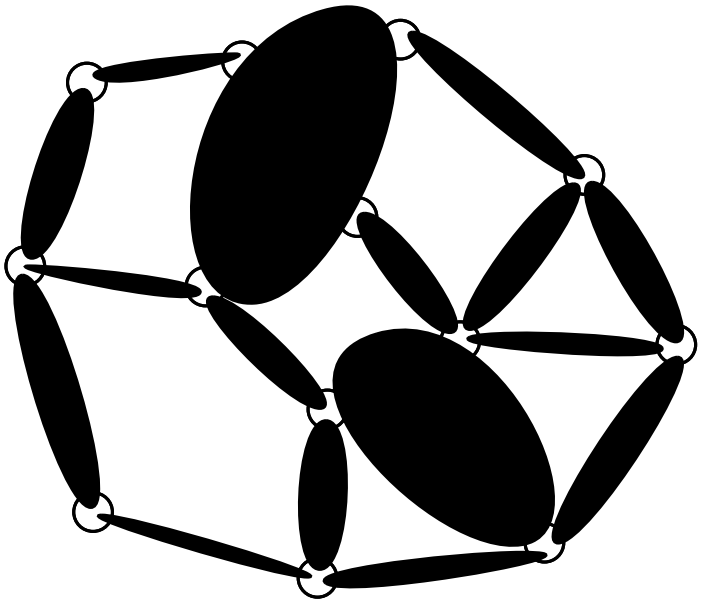

Figure 5.1: A hypergraph with two hyperedges.

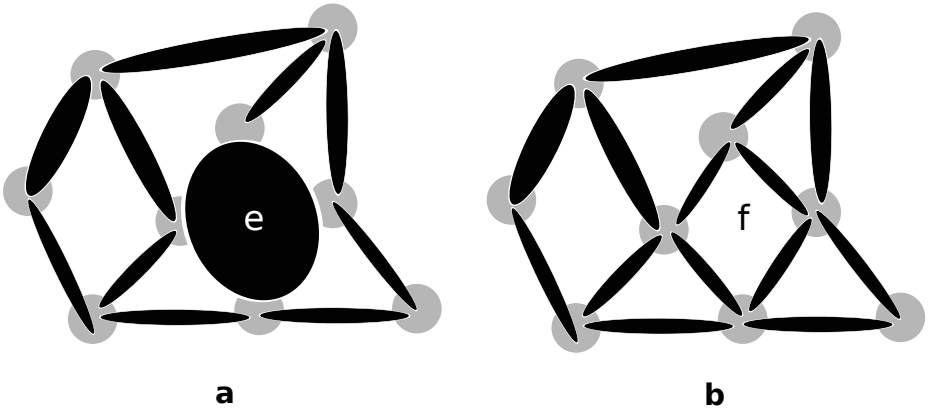

Figure 5.2: Hollowing a hyperedge e of a hypergraph and the resulting face f.

Now we are ready to give the formal description of the GT Algorithm. We first introduce two technical propositions from Gu and Tamaki's result [32] which are useful for bounding the branchwidth in the algorithm.

Proposition 5.2.1. *Let G be a hypergraph, (A, B) a separation of G such that $\partial(A) \notin A$ and $\partial(B) \notin B$, and T_A and T_B branch-decompositions of $G|B$ and $G|A$ respectively. Then $T_A + T_B$ is a branch-decomposition of G and its width is $\max\{k_A, k_B\}$ where k_A is the width of T_A and k_B is the width of T_B.*

Proposition 5.2.2. *Let G be a plane hypergraph and $g \geq 2$, $h \geq 1$ be integers. Suppose every hyperedge $e \in E(G)$ has order $\leq g$ and there is an hyperedge $e_0 \in E(G)$ such that for every vertex $v \in V(G)$, there is a vertex $v' \in V(G)$ incident to e_0 with $\mathrm{dist}(v, v') \leq h$. Then $\mathrm{bw}(G) \leq g + 2h$.*

Proposition 5.2.1 is used for constructing branch-decomposition through a series of recursions. Proposition 5.2.2 is a special case of the celebrated rat-catching characterization of the branchwidth of planar graphs due to Seymour and Thomas [48].

Theorem 5.2.1. *[32] For a planar graph G, let $g \geq 3$ and $h \geq 1$ be integers. G has either a minor isomorphic to a $(g \times h)$-cylinder or a branch-decomposition of width at most $g+2h-3$.*

Gu and Tamaki's proof of Theorem 5.2.1 is constructive, which implies an algorithm (the GT Algorithm). Given a planar graph G and two integers $g \geq 3$ and $h \geq 1$, the GT Algorithm finds either a $(g \times h)$-cylinder minor or a branch-decomposition of width at most $g + 2h - 3$. Since a grid minor can be obtained from a cylinder minor by removing some edges, we use cylinder minors as grid minors here. The contour technique described in Section 4.2 is used in this algorithm. We introduce their algorithm as follows.

Initially, we choose an edge e_0 and let the hyperedge $E_0 = e_0$. We denote by $V(E_0)$ the set of vertices E_0 incident to (i.e., $V(E_0) = \bigcup_{e \in E_0} V(e)$). Since the degree of E_0 is 2, we choose another edge e' that is not in E_0 but is incident to some vertex in $V(E_0)$, and add it into E_0. We repeat this process until $|V(E_0)| = g$. Let $G' = G|E_0$ be the plane graph obtained by hollowing hyperedge E_0 in G and this results in a face r_0 (since E_0 is the only hyperedge in G, G' is a graph). Let c_0 be the cycle of G' bounding r_0. Then we compute $\mathrm{cont}(r_0, h-1)$ which, by Proposition 4.2.1, consists of edge-disjoint cycles $c_1, c_2, ..., c_m$.

If, for some $1 \leq i \leq m$, the size of minimum vertex-cut that separates c_0 from c_i is at least g, then, according to Proposition 4.2.2, there is a minor of G isomorphic to a $(g \times h)$-cylinder and we are done.

Otherwise, for every $1 \leq i \leq m$, the size of a minimum vertex-cut that separates c_0 from c_i is smaller than g. Then for each i, we choose the minimum cut cut_i that separates c_0 form c_i and is nearest to c_i. For each cut cut_i, we find a noose ν_i corresponding to cut_i with the orientation such that c_0 is in $R(\nu_i)$. We then get $G'|\nu_i$ by replacing the connected component of G' from c_0 to ν_i with a hyperedge E'_0 and recursively do the above procedure by treating E'_0 as E_0. At last, if no cylinder minor is found, due to Proposition 5.2.1 and Proposition 5.2.2, we claim that the branchwidth of G is at most $g + 2h - 3$.

The problem of finding a minimum vertex-cut in a planar graph can be reduced to the max-flow problem in linear time [34]. The time complexity of the GT Algorithm is $O(n^2)$. The proof of the correctness of this algorithm is complicated. Readers may refer to the original paper [32] for details.

5.3 Computational results

5.3.1 Comparison between the BGK Algorithm and the GT Algorithm

Bodlaender et al. [5] present an extensive computational study on the BGK Algorithm for the largest grid minor problem. They test their algorithm on planar graphs from TSPLIB [43]. We implement the GT Algorithm to find the grid minors (implied by cylinder minors) and compare the sizes of the grid minors found by these two algorithms.

The same computer used for testing the exact algorithm in Chapter 4 is used for testing the GT Algorithm. The algorithm is implemented with C++.

The results are tabulated in Table 5.1, where "bw" is the branchwidth of the graph, "gm-BGK" is the size of $(g \times g)$-grid minor found by the BGK Algorithm, "cm-GT" is the size of $(g \times h)$-grid found by the GT Algorithm, and "cm-GT time" indicates the time for finding it (in seconds). If we restrict $g = h$, the size of $(g \times g)$-grid minor found by the GT Algorithm is shown in column "gm-GT", and "gm-GT time" indicates the time for finding it (in seconds).

For the size of $(g \times g)$-grid minor, Table 5.1 shows that, in only two (d1665 and tsp225) out of 20 instances of graphs, the g obtained by the GT Algorithm is smaller than that obtained by the BGK Algorithm. According to Bodlaender et al. [5], the computer they use for the computational study on the BGK Algorithm has the same processor with the computer we use. Due to the difference of implementation details, we don't compare the running times of the two implementations. However, from Table 5.1, it can be shown that

Table 5.1: Comparison of the computational results between the BGK Algorithm and the GT Algorithm.

graphs	# of edges	bw	gm-BGK	gm-GT	gm-GT time	cm-GT	cm-GT time
eil51	140	8	4	4	0.06	7×4	0.05
lin105	292	8	5	6	0.1	7×6	0.1
ch130	377	10	5	6	0.3	9×5	0.27
pr144	393	9	5	5	0.25	9×4	0.28
kroB150	436	10	7	7	0.42	11×5	0.16
tsp225	622	12	9	7	0.22	14×5	0.21
pr226	586	7	5	5	0.2	11×3	0.62
a280	788	13	6	6	0.31	15×5	0.31
pr299	864	11	6	7	0.49	11×6	0.23
rd400	1183	17	10	10	1.3	17×8	0.89
pcb442	1286	17	9	9	0.86	15×8	1.16
u574	1708	17	11	13	2.04	15×12	2.65
p654	1806	10	7	7	0.82	7×7	0.82
d657	1958	22	11	12	4.9	24×8	2.04
pr1002	2972	21	12	13	15.39	20×10	8.57
rl1323	3950	22	13	15	17.09	17×14	14.28
d1655	4890	29	19	18	15.43	22×16	15.79
rl1889	5631	22	14	16	23.37	18×15	25.09
u2152	6312	31	22	22	33.98	22×23	36.57
pr2392	7125	29	16	18	21.76	18×19	30.14

the GT Algorithm is practical even for large planar graphs with thousands of edges: this algorithm is able to find large grid minors of a large planar graph within a few minutes. Since the algorithm starts with a random edge, the running time varies with each execution. We run the algorithm for two to three times and choose the one that yields the best grid minor. It is worth mentioning that, the GT Algorithm is more flexible than the BGK Algorithm, since it can compute a $(g \times h)$-grid minor without the constraint that $g = h$. Therefore, we conclude that the GT Algorithm is more practical and suitable for the computational study of bidimensionality theory based algorithms.

5.3.2 Quality of the cylinder minors computed by the GT Algorithm

The exact algorithm introduced in Chapter 4 provides a tool for evaluating how close the cylinder minors found by approximation algorithms are to the largest ones.

Table 5.2 shows the comparison between the sizes of cylinder minors found by the GT Algorithm and those found by the exact algorithm which is described in Chapter 4. In the table, "cm-GT" is the size of the cylinder minor found by the GT Algorithm, while "cm-optimal" is the size of the cylinder minor found by the exact algorithm. Similarly to Table 4.1, entries in Table 5.2 labeled with "*" indicate that the results are not known to be optimal as explained in Section 4.3.

Table 5.2: Closeness between the size of the optimal cylinder minors and the size of the cylinder minors found by the GT Algorithm.

graphs	# of edges	cm-GT	cm-optimal
eil51	140	7×4	7×4
eil76	215	8×5	8×5
pr76	218	8×5	8×6
rat99	279	7×6	7×6
rd100	286	7×6	7×6*
kroB100	284	8×5	8×6*
lin105	292	7×6	7×7
ch130	377	9×5	9×6*
pr144	393	9×4	9×4*

For some instances (pr76, kroB100, lin105, and ch130), the cylinder minors computed by the GT Algorithm are smaller than those computed by the exact algorithm, but are very

close to the optimal ones. For other instances, the GT Algorithm computes cylinder minors as large as the exact solutions.

For planar graphs of fewer than 100 vertices, the computational results suggest that the quality of cylinder minors found by the GT Algorithm is very close to that found by the exact algorithm. Therefore, we expect that for large planar graphs, the GT Algorithm still produces near-optimal solutions.

Chapter 6

Computational Study for the Longest Path Problem

6.1 Non-crossing property

The branch-decomposition based dynamic programming algorithm for the longest path problem in a general graph G is described in Section 3.2. Recall that in each dynamic programming step, the number of possible characters of a candidate set of m vertices equals the number of partitions of m elements. A partial solution in each step is a set of vertex-disjoint paths in some subgraph of G. If we draw a partial solution on a plane, these vertex-disjoint paths might cross, since G is not necessarily planar. If G is a plane graph, these vertex-disjoint paths on the plane do not cross. Above is an intuitive description of the *non-crossing property*, which reduces the number of states in the dynamic programming algorithm for planar graphs significantly.

The non-crossing property [25] is the key to speedup the dynamic programming algorithm in planar graphs as compared to non-planar graphs. Placing n vertices on a cycle, a *non-crossing matching* can be visualized as connecting matched vertices with chords inside the cycle, without crossing. Given a counter-clockwise (or clockwise) order of the vertices along the cycle starting with an arbitrary vertex, we color the first encountered vertex of a pair from the matching with "[", and the other vertex of the pair with "]" (see Figure 6.1). If the matching is non-crossing, it can be identified by a specific coloring. The number of

non-crossing matchings of l vertices is the $\frac{l}{2}$-th Catalan number [25, 37]:

$$M(l) = CN(\frac{l}{2}) \sim \frac{2^l}{\sqrt{\pi}(\frac{l}{2})^{1.5}} \approx 2^l. \tag{6.1}$$

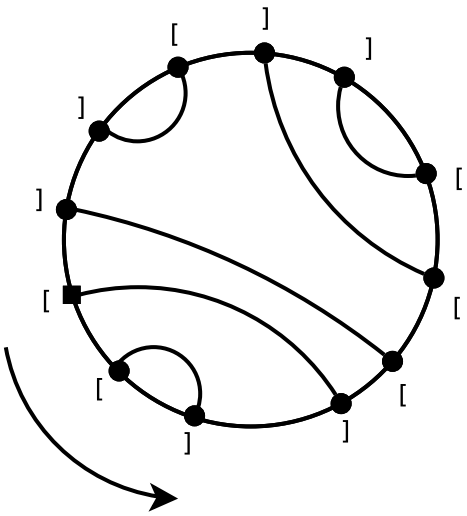

Figure 6.1: A non-crossing matching of 12 vertices on a cycle and the coloring. The order is counter-clockwise starting with the square vertex.

6.2 The DPBF Algorithm

Below we describe the branch-decomposition based dynamic programming algorithm by Dorn et al. (the DPBF Algorithm). Readers may refer to the original paper [25] for more details. We first compute an optimal sc-decomposition T of a planar graph G. Then we do dynamic programming based on T. We first convert an sc-decomposition T to a rooted binary tree by replacing an arbitrary link $e = \{u, v\}$ of T with three links $\{u, s\}$, $\{s, v\}$ and $\{s, r\}$. The new vertex r is called the *root* of T. Let $\mathrm{mid}(\{s, r\}) = \emptyset$. Removing a link e in T leaves two subtrees of T. We call the subtree containing the root r the *upper part* and the other the *lower part*. We denote by G_e the subgraph of G induced by the edges associated with the leaves of the lower part of e, and by $\overline{G_e}$ the subgraph of G induced by edges associated with the leaves of the upper part of e. Let O_e be a G-noose separating G_e from $\overline{G_e}$. Notice that the non-crossing property applies to $\partial(E(G_e))$. Let $V(O_e)$ denote the

vertices on O_e. Therefore $V(O_e)$ is the middle set of the separation induced by e. For each internal vertex v of T, there exists one adjacent link e_p on the path from v to r. The two adjacent links towards the leaves are denoted by e_l and e_r respectively.

Let O_p, O_l, and O_r be the noose corresponding to the separations induced by the links e_p, e_l, and e_r respectively. We partition the set $V(O_p) \cup V(O_l) \cup V(O_r)$ into four subsets:

- $X_1 = V(O_l) \cap V(O_p) \setminus V(O_r)$
- $X_2 = V(O_r) \cap V(O_p) \setminus V(O_l)$
- $X_3 = V(O_p) \cap V(O_l) \cap V(O_r)$
- $X_4 = V(O_l) \cap V(O_r) \setminus V(O_p)$

We define a labeling $P_e : E(G_e) \to \{0, 1\}$ on each edge of G_e. Let G_{P_e} be the subgraph induced by the edges with label "1". For each vertex v in G_e, let $\deg_{P_e}(v)$ denote the sum of the labels of the edges incident to v. We call P_e a valid partial solution of the longest path problem if the following conditions are satisfied.

1. There are at most two vertices $v \in V(G_e) \setminus V(O_e)$ with $\deg_{P_e}(v) = 1$ and for other vertices $v \in V(G_e) \setminus V(O_e)$, $\deg_{P_e}(v) = 2$.
2. There are at most two connected components of G_{P_e} such that in the connected component there is only one vertex in $V(O_e)$ with $\deg_{P_e}(v) = 1$. Each of the connected components has exactly two vertices in $V(O_e)$ with $\deg_{P_e}(v) = 1$. All other vertices of G_{P_e} have $\deg_{P_e}(v) = 2$.

Intuitively, for a partial solution P_e, each connected component of G_{P_e} is a path in G_{P_e}. The connected components of G_{P_e} form a set of vertex-disjoint paths. We define the *length* of a partial solution P_e to be the number of edges with label "1" in G_{P_e} (i.e., the sum of the lengths of these vertex-disjoint paths in G_{p_e}).

Let π be the counter-clockwise (or clockwise) order starting with an arbitrary vertex in the middle set $V(O_e)$ of the separation induced by e. We color each vertex $v \in V(O_e)$ in the order π with one of the following five colors.

- "0": v is not in any connected component of G_{P_e}.
- "$1_{[}$": v is the first encountered end vertex (in the order π) in some connected component P of G_{P_e}, and both end vertices of P are in $V(O_e)$.

- "$1_]$": v is the second encountered end vertex (in the order π) in some connected component P of G_{P_e} (i.e., another end vertex of P is already colored with "$1_[$"), and both end vertices of P are in $V(O_e)$.
- "1": v is the first encountered end vertex (in the order π) in some connected component P of G_{P_e}, and P has only one end vertex in $V(O_e)$.
- "2": v is in some connected component P of G_{P_e} and this vertex is not an end vertex of P.

For a link e in T, the state (coloring) of dynamic programming is an ordered l-tuple $t_e = (c_1, c_2, ..., c_l)$, where l is the order of the separation induced by e. Each c_i $(1 \leq i \leq l)$ takes one of the five colors 0, 1, $1_[$, $1_]$, 2. There is a variable L corresponding to each state t_e which keeps the maximum length of the valid partial solutions that can be represented by this state. For a state t_e, we denote by $t_e[v]$ for $v \in \text{mid}(e)$ the color of v in this state. We also define the *numerical value* of each color: $\text{num}(0) = 0$, $\text{num}(1_[) = 1$, $\text{num}(1_]) = 1$, $\text{num}(1) = 1$ and $\text{num}(2) = 2$,.

We perform dynamic programming on the links of T from the leaves towards the root. For each leaf vertex of T, we set the color state $(0, 0)$ with length $L = 0$ and $(1_[, 1_])$ with length $L = 1$. For an internal vertex v in T, we say that the state t_{e_p} is formed by the state t_{e_r} and t_{e_l} if:

- For $v \in X_1$, $t_{e_p}[v] = t_{e_l}[v]$.
- For $v \in X_2$, $t_{e_p}[v] = t_{e_r}[v]$.
- For $v \in X_3 \cup X_4$, $\text{num}(t_{e_p}[v]) + \text{num}(t_{e_r}[v]) \leq 2$.

If a connected component of a partial solution P_e has only one end vertex in $\text{mid}(e)$, we call this connected component an "only-end segment" (i.e., the end vertex is colored with "1"). If a connected component has no vertex in $\text{mid}(e)$, we call this connected component a "lonely segment". A state is valid if the partial solution corresponding to it has at most two only-end segments. Moreover, during the dynamic program, the number of lonely segments cannot exceed one.

Dorn et al. [25] show that this algorithm can find a longest path in a planar graph in $O(2^{7.223\sqrt{n}} + n^3)$ time. The running time of this algorithm depends on the number of states in each dynamic programming step. To analyze the time complexity, first notice that an

sc-decomposition T of a planar graph G can be constructed in $O(n^3)$ [33, 48]. For a link e of T, which induces a separation of order l, without loss of generality, we make the following assumptions: l is an even integer, $|V(O_l)| = |V(O_r)| = |V(O_p)| = l$ and $X_3 = \emptyset$ (i.e., $|X_1| = |X_2| = |X_4| = l/2$).

We first assume that there is no only-end segment (i.e., no vertex is colored with "1"). For $v \in X_4$, if the colors of v on both sides (i.e., $t_{e_l}[v]$ and $t_{e_r}[v]$) have numerical value 1, we call this vertex fixed. Otherwise, we call it unfixed. Let $Q(l, m)$ be the number of colorings of l vertices where m vertices in X_4 are fixed. The freedom is in the $l/2$ vertices in X_1 and the $l/2$ vertices in X_2. Therefore we obtain:

$$Q(l, m) = \sum_{i=0}^{l/2} \binom{\frac{l}{2}}{i} 2^{\frac{l}{2}-i} M(i + m). \tag{6.2}$$

Formula (6.2) is a summation over the number of vertices colored with "$1_[$" and "$1_]$" in X_1 and X_2 respectively. The term $\binom{\frac{l}{2}}{i}$ counts the possible vertices to be colored with "$1_[$" and "$1_]$". The $2^{\frac{l}{2}-i}$ counts the possible colors "0" and "2" for the remaining $l/2 - i$ vertices. $M(i + m)$ is the number of non-crossing matchings of the vertices colored with "$1_[$" and "$1_]$". By (6.1) and (6.2),

$$Q(l, m) = O\left(\sum_{i=0}^{l/2} \binom{\frac{l}{2}}{i} 2^{\frac{l}{2}-i} 2^{i+m}\right) = O(2^{l+m}). \tag{6.3}$$

Let $C(l)$ be the number of possible colorings for the separation induced by e. With (6.3), we get:

$$C(l) = \sum_{i=0}^{l/2} \binom{\frac{l}{2}}{i} 3^{\frac{l}{2}-i} (Q(l, i))^2 \tag{6.4}$$

$$= O\left(\sum_{i=0}^{l/2} \binom{\frac{l}{2}}{i} 3^{\frac{l}{2}-i} 2^{2l} 2^{2i}\right) \tag{6.5}$$

$$= O\left((4\sqrt{3})^l \sum_{i=0}^{l/2} \binom{\frac{l}{2}}{i} (4/3)^i\right) \tag{6.6}$$

$$= O((4\sqrt{7})^l) = O(2^{2.304l}). \tag{6.7}$$

Formula (6.4) is a summation over the number of fixed vertices in X_4. Since there are three ways to color an unfixed vertex $v \in X_4$: (1) $t_{e_l}[v] = 0$ and $t_{e_r}[v] = 2$, (2) $t_{e_l}[v] = 2$ and

$t_{e_r}[v] = 0$, (3) $t_{e_l}[v] = 0$ and $t_{e_r}[v] = 0$, the $3^{\frac{l}{2}-i}$ counts the number of possible ways for coloring these unfixed vertices. The $Q(l, i)$ is squared because the situations for coloring fixed vertices with colors of numerical value 1 on both sides are symmetric.

Now we allow the number of vertices colored with "1" to be up to two. This brings a factor $\binom{l}{2} = O((\text{bw}(G))^2) = O(n)$ to the polynomial part of $C(l)$ and does not affect the exponential behavior. Since we can check to see whether a coloring is valid in linear time in l and we record the partial solution when computing each coloring in linear time in n, the overall running time is $O(n2^{3.404l}ln + n^3) = O(2^{3.404\text{bw}(G)}\text{bw}(G)n^2 + n^3)$. According to [27], $\text{bw}(G) \leq \sqrt{4.5n} \leq 2.122\sqrt{n}$, where n is the number of vertices of G. Therefore we conclude that the time complexity of this algorithm is $O(2^{7.223\sqrt{n}}n^{5/2} + n^3)$.

We use the size of a cylinder minor instead of a grid minor to analyze the time complexity of the algorithms for solving the parameterized planar longest path problem (planar k-longest path problem). Gu and Tamaki show that $\text{bw}(G) \leq 2\text{cm}(G)$ for a planar graph G [32] and the GT Algorithm actually computes a $(g \times h)$-cylinder minor of G. If $\text{bw}(G) \geq 2\sqrt{2(k+1)}$ for some parameter value k, it implies that G has a $(\sqrt{2(k+1)} \times \lceil\sqrt{2(k+1)}/2\rceil)$-cylinder minor. Thus we conclude the length of the longest path is at least k. When $\text{bw}(G) < 2\sqrt{2(k+1)}$, we compute the longest path on G using the DPBF Algorithm in $O(2^{3.404\times2\sqrt{2k}}n^{5/2} + n^3) = O(2^{9.628\sqrt{k}}n^{5/2} + n^3)$ time.

There are $O(l4^l)$ states in each dynamic programming step on a link e of T, where l is the order of the separation induced by e. Therefore the potential maximum memory usage is $O(4^{\text{bw}(G)})$. In order to reduce memory usage, we use a two-level mapping table (see Figure 6.2) to keep the states. The first level is indexed by numerical values of colorings. Each numerical value is mapped to a second level mapping table which contains a block of colorings and is indexed by actual colorings whose numerical values are their key in the first level mapping table. We only keep valid states in the mapping table. In each dynamic programming step, we first check to see whether the numerical values satisfy $\forall v \in X_3 \cup X_4$, $\text{num}(t_{e_l}[v]) + \text{num}(t_{e_r}[v]) \leq 2$. If the condition is satisfied, we check each pair of the actual colorings inside the second level blocks. Otherwise, we omit the coloring blocks of these two numerical values.

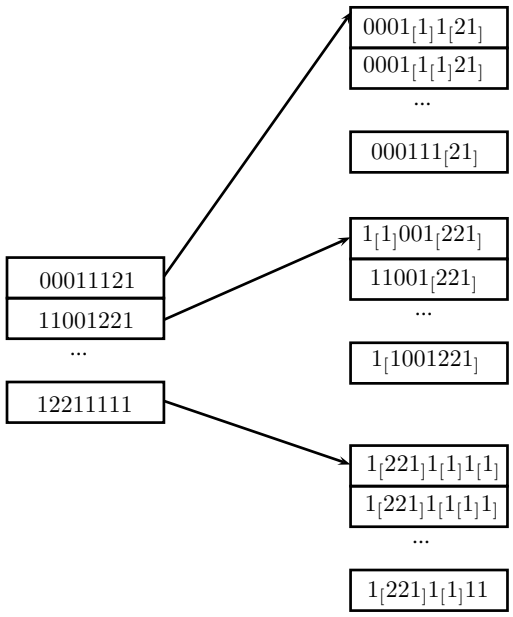

Figure 6.2: An instance of the two-level mapping table.

6.3 Computational results

6.3.1 Computing the exact longest path

We test the algorithm for solving the planar longest path problem on the following classes of graphs. Class I is a set of random maximal planar graphs generated by LEDA [39]. Class II is taken from TSPLIB [43]. Class III contains the triangulation graphs generated by LEDA. Class IV consists of Gabriel graphs generated with the points uniformly distributed in a 2-D plane. Class V is a set of random planar graphs generated by the PIGALE library [12]. Class VI consists of intersection graphs generated by LEDA. These classes of graphs are representative of planar graphs and are widely used in research on related problems.

The computer we use has an AMD Athlon 64 X2 Dual Core 4600+ (2.4 GHz) CPU and 3 GByte RAM. The operating system is SUSE Linux 10.2. The algorithm is implemented with C++. We use a different computer from the one we described in Chapter 4 since this computer is faster and it is suitable for programs which are likely to run for several hours.

The results are shown in Table 6.1. We use the programs by Bian and Gu [3] to compute the optimal branch-decomposition. In Table 6.1, "bw" is the branchwidth and "bw-time"

shows the time (in seconds) for computing the optimal branch-decomposition. It is shown that the time grows as the size of the graphs increases. The column "lp" indicates the length of the longest path returned by the algorithm. For graphs in TSPLIB (Class II), the lengths show that the longest paths are Hamiltonian paths as expected. The column "lp-time" indicates the time (in seconds) for computing the longest path using the DPBF Algorithm, and "total-time" is the total running time (computing optimal branch-decomposition and computing longest path). The column "max-mem" indicates the memory space (in MByte) used by the algorithm. As shown in Section 4, the running time is exponential in the branchwidth. Graphs in Class I have small branchwidth (at most 4 [38]). The algorithm can compute the longest path in large graphs of more than ten thousand edges in Class I in a few minutes. Branchwidth grows very quickly for graphs in Classes II and IV. It is shown in Table 6.1 that for graphs of branchwidth 10, the algorithm is able to compute the longest path in a few hours. For graphs in Classes III and VI, branchwidth grows relatively slowly. The largest size of graphs in Classes III and VI the implementation of this algorithm can handle is therefore larger than that in Classes II and IV. For Classes II and IV, the implementation of the algorithm can handle graphs of a few hundred edges, while for Classes III and VI, the implementation of the algorithm can handle graphs of a few thousand edges. Branchwidth does not grow for graphs in Class V. Therefore the implementation of the algorithm can handle larger graphs in Class V than in Classes II and IV.

Memory usage is also exponential in branchwidth. This is a bottleneck in the DPBF Algorithm. It is shown in Table 6.1 that the algorithm is able to compute the longest path in planar graphs with branchwidth ≤ 10 with 3 GByte memory.

6.3.2 Lower bounds on the longest path problem

When the branchwidth of a planar graph G is large, using the DPBF Algorithm to compute the longest path of G is not practical. As discussed in Chapter 3, bidimensionality theory provides a method to compute a lower bound on the problem.

We use the GT Algorithm [32] to compute the grid minor of planar graphs. A grid minor implies a lower bound on the length of the longest path. We test the algorithm on Classes I to V which are described in Section 6.3.1. The same computer used for the grid minor study in Chapter 4 is used for this study.

The results are tabulated in Table 6.2. In the table, "cm" is the size of the cylinder minor found by the GT Algorithm and "cm-time" is the time (in seconds) for finding it. A

Table 6.1: Computational results of the DPBF Algorithm for planar longest path problem. For graphs labeled with "*", 3 GByte memory is not enough for the algorithm.

class	graphs	# of edges	bw	bw-time	lp	lp-time	total-time	max-mem
I	max1000	2994	4	68.05	731	14.14	82.19	15.32
	max2000	5994	4	301.68	1436	50.7	352.38	28.8
	max3000	8994	4	1095.91	1807	111.84	1207.75	45.92
	max4000	11994	4	1856.6	2557	196.67	2053.27	72.65
	max5000	14994	4	3138.08	3164	303.7	3441.78	94.12
II	eil51	140	8	0.11	50	145.3	145.41	38.23
	lin105	292	8	2.48	104	654.44	656.92	90.47
	pr144	393	9	0.52	143	2742.86	2743.38	306.11
	kroB150	436	10	0.83	149	91475.42	91476.25	2014.36
	pr226	586	7	1.56	225	96.53	98.09	45
	ch130	377	10	0.54	129	38787.61	38788.15	998.6
III	tri100	288	6	0.65	99	6.91	7.56	9.7
	tri500	1480	7	24.39	499	308.37	332.76	115.22
	tri700	2074	7	95.47	699	505.72	601.19	127.26
	tri1000	1491	8	77.97	999	4302.2	4380.17	864.01
	tri2000	5977	8	638.36	1999	21440.18	22078.54	1545.81
	tri3000*	8976	8	2559.34	-	-	-	-
IV	gab50	88	7	0.22	49	0.11	0.33	0.14
	gab100	182	7	0.32	99	11.88	12.2	17.52
	gab200	366	8	0.97	199	407.57	408.54	108.35
	gab300	552	10	1.51	297	24714.59	24716.1	1449.08
V	p153	248	4	0.97	131	0.33	1.3	5.13
	p257	433	5	1.73	226	1.94	3.67	8.66
	p495	852	5	11.66	426	4.97	16.63	14.74
	p855	1434	6	21.06	726	28.05	49.11	59.03
	p1277	2128	9	61.98	1087	10074.09	10135.07	2308.92
	p2009	3369	7	232.45	1670	639.93	639.93	309.85
	p3586	6080	8	2583.52	2982	8224.52	8224.52	2311.43
VI	rand100	121	3	0.21	45	0.1	0.31	0.14
	rand500	709	6	0.94	237	4.47	5.41	20.81
	rand700	1037	6	1.46	291	7.9	9.36	24.03
	rand1000	1512	7	2.6	715	96.43	99.03	141.04
	rand2000	3247	8	17.95	1591	1421.82	1439.77	525.83
	rand3000*	4943	10	55.95	-	-	-	-

$(g \times h)$-grid minor of G implies a path of length $\geq g \times h - 1$, which is a lower bound on the length of the longest path of G. The formula based lower bound is given in terms of branchwidth. Since $\text{bw}(G) \leq 2\text{cm}(G)$, we get $\text{cm}(G) \geq \frac{\text{bw}(G)}{2}$. Thus the lower bound given by branchwidth is $\frac{(\text{bw}(G))^2}{8} - 1$ (this value is given in the column of "fmla. LB" in Table 6.2). In practice, if we compute a large cylinder/grid minor of G, a computation based lower bound can be obtained directly by the size of the cylinder/grid minor. In this table, the column "cmpt. LB" shows the computation based lower bound on the length of the longest path implied by the cylinder. The GT Algorithm starts from an arbitrary edge. Therefore the results differ with every execution. We run the algorithm for each graph several times and select the largest grid minor it found.

By Theorem 5.2.1, given integers $g \geq 3$ and $h \geq 1$, if no cylinder minor is found, the upper bound of branchwidth is $g + 2h - 3$. Therefore, we try the combinations of g and h with g approximately equal to $2h$ in order to maximize $g \times h$. For graphs in Class I, the branchwidth is small. Thus we test g and h with $2g$ approximately equal to h. Graphs in Class II have larger branchwidth, and the formula based lower bound is relatively larger. We get larger cylinder minor in these graphs. The characteristics about the formula based lower bound of graphs in Classes III, VI, and V are similar.

For all tested graphs, the computation based lower bounds obtained by the algorithm are much better than the formula based lower bounds. This is because the grid minor found in practice is larger than the formula based one. Theorem 5.2.1 implies $\text{bw}(G) \leq 3\text{gm}(G)$. There is a theoretical gap between branchwidth and the size of a grid minor with a factor 3. In practice, this gap is smaller. For some graphs such as lin105, tri100, gab200, gab300, and p855, the factor achieves 2. For other graphs in Classes II, III, IV and V, the factor is very close to 2.

The results suggest that the computation based lower bounds obtained by the GT Algorithm are much better than the formula based lower bounds for the longest path problem. Moreover, the GT Algorithm is practical. For large planar graphs with thousands of edges, a lower bound on the longest path problem can be obtained by this algorithm within a few minutes.

Table 6.2: Computational results of the GT Algorithm and the comparison between the computation based lower bounds and the formula based ones.

class	graphs	# of edges	bw	cm	cm-time	cmpt. LB	fmla. LB
I	max1000	2994	4	3×7	1.44	20	1
	max2000	5994	4	3×7	5.03	20	1
	max3000	8994	4	3×7	10.21	20	1
	max4000	11994	4	3×7	18.89	20	1
	max5000	14994	4	3×7	29.61	20	1
II	lin105	292	8	8×4	0.09	31	7
	kroB150	436	10	11×5	0.16	54	11
	pr226	586	7	8×3	0.66	23	5
	ch130	377	10	9×5	0.27	44	11
	pr1002	2972	21	20×10	8.57	199	54
	d1655	4890	29	24×12	17.73	287	104
	u2152	6312	31	24×13	31.96	311	119
III	tri100	288	6	6×3	0.55	17	3
	tri500	1480	7	6×3	4.93	17	5
	tri1000	1491	8	6×5	27.58	29	7
	tri2000	5977	8	6×5	142.74	35	7
	tri3000	8976	8	6×5	23.03	29	7
IV	gab100	182	7	8×3	0.1	23	5
	gab200	366	8	8×4	0.29	31	7
	gab300	552	10	10×5	0.32	49	11
	gab700	1302	15	10×8	1.82	79	27
V	p257	433	5	4×4	0.21	15	2
	p855	1434	6	6×3	4.6	17	3
	p1277	2128	9	9×4	16.09	35	9
	p2009	3369	7	7×3	17.17	20	5
	p3586	6080	8	6×5	139.31	29	7

Chapter 7

Conclusion and Future Work

7.1 Summary of contributions

In theory, bidimensionality theory and branch-decomposition based dynamic programming algorithms provide a general framework for developing subexponential fixed parameter algorithms to solve some NP-hard problems, such as planar dominating set, vertex cover, planar longest path, etc.. However, before the work of this thesis, whether this framework is practical was not known because of the lack of tools for computing large grid minors.

We design and implement an exact algorithm for computing the largest cylinder minor in planar graphs. However, there is no polynomial bound on the running time of this algorithm. Computational results show that the implementation of this algorithm can handle small planar graphs of less than 100 vertices with 2 GByte memory. Since the exact algorithm is not practical, approximation algorithms attract more attention and interests. The exact algorithm provides a tool to evaluate the performance of the approximation algorithms for computing large cylinder minors.

Bodlaender et al. give a 4-approximation algorithm (the BGK Algorithm) for the largest grid minor problem in planar graphs. They also study the computational performance of the algorithm and the results show that their algorithm is efficient in practice.

Gu and Tamaki improve the linear relationship between the branchwidth and the size of the largest grid minor from $\mathrm{bw}(G) \leq 4\mathrm{gm}(G)$ [44] to $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ [32]. Their proof of the bound is constructive, which implies a 3-approximation algorithm (the GT Algorithm) for the largest grid minor problem in planar graphs. In this thesis, we implement and study this algorithm. In the comparison between the GT Algorithm and the BGK Algorithm, we

show that the grid minors found by the GT Algorithm are larger than the grid minors found by the BGK Algorithm. For small instances of planar graphs, the sizes of cylinder minors computed by the GT Algorithm and those computed by the exact algorithm are very close. Therefore we could expect that for large instances of planar graphs, the GT algorithm also finds near-optimal cylinder minors. Moreover, we show that the GT Algorithm is practical even when handling large planar graphs with 15,000 edges.

For the longest path problem, if a graph G contains a $(g \times h)$-grid/cylinder minor, it implies that the lower bound on the length of the longest path is $g \times h - 1$. Naturally, $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ (or $\mathrm{bw}(G) \leq 2\mathrm{cm}(G)$) implies a formula based lower bound on the length of the longest path in terms of branchwidth. The grid minor found by the GT Algorithm implies a computation based lower bound. We compare the formula based and computation based bounds. The computational results show that the computation based lower bounds on the planar longest path problem obtained by the GT Algorithm are much better than the formula based ones.

We also study the DPBF Algorithm for the planar longest path problem. We evaluate their algorithm on extensive classes of planar graphs. The algorithm first computes an optimal branch-decomposition for a graph and then does dynamic programming based on it. The running time and memory usage are exponential in the branchwidth. To save memory, we use a two-level mapping table to store only valid states in each dynamic programming step. The computational results show that the algorithm can compute the longest path for planar graphs of branchwidth at most 10 with a 3 GHz processor and 3 GByte memory.

7.2 Future work

The problem of computing the largest grid minor in general graphs is NP-hard [31]. Whether it remains NP-hard in planar graphs is still an open problem. One direction of future research is the proving of its NP-completeness or seeking a polynomial time algorithm for the largest grid minor problem in planar graphs.

For every positive integer h, the size of the largest grid minor $\mathrm{gm}(C_{2h,h})$ of a cylinder $C_{2h,h}$ is h [32]. Moreover, its branchwidth $\mathrm{bw}(C_{2h,h})$ is $2h$. This implies that it is impossible to improve the coefficient 3 in $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ to below 2. However the tightness of the relationship $\mathrm{bw}(G) \leq 3\mathrm{gm}(G)$ is not known. Future research may focus on finding a tighter bound, which implies larger grid minors, or asserting that this bound is tight.

Bibliography

[1] K. Asdre and S.D. Nikolopoulos. The 1-fixed-endpoint path cover problem is polynomial on interval graphs. *Algorithmica*, pages 1–32, 2009.

[2] D. Berend and T. Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2), 2010.

[3] Z. Bian and Q.P. Gu. Computing branch decomposition of large planar graphs. *Experimental Algorithms*, pages 87–100, 2008.

[4] Z. Bian, Q.P. Gu, M. Marzban, H. Tamaki, and Y. Yoshitake. Empirical study on branchwidth and branch decomposition of planar graphs. In *Proc. of the 9th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX08)*, pages 152–165, 2008.

[5] H.L. Bodlaender, A. Grigoriev, and A.M.C.A. Koster. Treewidth lower bounds with brambles. *Algorithms–ESA 2005*, pages 391–402, 2005.

[6] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel. On the number of cycles in planar graphs. *Computing and Combinatorics*, pages 97–107, 2007.

[7] T.H. Byers and M.S. Waterman. Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research*, 32(6):1381–1384, 1984.

[8] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[9] P. Damaschke. The Hamiltonian circuit problem for circle graphs is NP-complete. *Information Processing Letters*, 32(1):1–2, 1989.

[10] P. Damaschke. Paths in interval graphs and circular arc graphs. *Discrete Mathematics*, 112(1-3):49–64, 1993.

[11] P. Damaschke, J.S. Deogun, D. Kratsch, and G. Steiner. Finding Hamiltonian paths in cocomparability graphs using the bump number algorithm. *Order*, 8(4):383–391, 1991.

[12] H. de Fraysseix and P.O. de Mendez. PIGALE-Public Implementation of a Graph Algorithm Library and Editor. *SourceForge project page http://sourceforge.net/projects/pigale*.

[13] E. Demaine, F. Fomin, M. Hajiaghayi, and D. Thilikos. Bidimensional parameters and local treewidth. *LATIN 2004: Theoretical Informatics*, pages 109–118, 2004.

[14] E.D. Demaine, F.V. Fomin, M. Hajiaghayi, and D.M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.

[15] E.D. Demaine and M.T. Hajiaghayi. Bidimensionality, map graphs, and grid minors. *Arxiv preprint cs/0502070*, 2005.

[16] E.D. Demaine and M.T. Hajiaghayi. Bidimensionality: New connections between fpt algorithms and ptass. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 590–601. Society for Industrial and Applied Mathematics, 2005.

[17] E.D. Demaine and M.T. Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 682–689. Society for Industrial and Applied Mathematics, 2005.

[18] E.D. Demaine and M.T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 2007.

[19] E.D. Demaine and M.T. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

[20] E.D. Demaine, M.T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. 2005.

[21] E.D. Demaine, M.T. Hajiaghayi, and D.M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM Journal on Discrete Mathematics*, 20(2):357–371, 2007.

[22] F. Dorn. Dynamic programming and fast matrix multiplication. *Algorithms–ESA 2006*, pages 280–291, 2006.

[23] F. Dorn, F.V. Fomin, and D.M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 631–640. Society for Industrial and Applied Mathematics, 2008.

[24] F. Dorn, F.V. Fomin, and D.M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

[25] F. Dorn, E. Penninkx, H.L. Bodlaender, and F.V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. *Algorithms–ESA 2005*, pages 95–106, 2005.

[26] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

[27] F.V. Fomin and D.M. Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51(1):53–81, 2006.

[28] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.

[29] M.C. Golumbic. *Algorithmic graph theory and perfect graphs*. North-Holland, 2004.

[30] T.F. Gonzalez. *Handbook of approximation algorithms and metaheurististics*. Chapman & Hall, 2007.

[31] Qian-Ping Gu and Hisao Tamaki. Manualscript.

[32] Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size, 2011. Submitted for publication.

[33] Q.P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in O (n 3) time. *ACM Transactions on Algorithms (TALG)*, 4(3):1–13, 2008.

[34] Q.P. Gu and H. Tamaki. Efficient reduction of vertex-disjoint Menger problem to edge-disjoint Menger problem in undirected planar graphs. Technical report, Technical Report, SFU-CMPT-TR 2009-11, 2009.

[35] Y.C. Hsu, S. Sun, and D.H.C. Du. Finding the longest simple path in cyclic combinational circuits. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference on*, pages 530–535. IEEE, 2002.

[36] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11:676, 1982.

[37] G. Kreweras. Sur les partition non croisées dun circle. *Discrete Mathematics*, 1:333–350, 1972.

[38] M. Marzban, Q.P. Gu, and X. Jia. Computational study on planar dominating set problem. *Theoretical Computer Science*, 410(52):5455–5466, 2009.

[39] K. Mehlhorn, S. Näher, and C. Uhrig. The LEDA platform for combinatorial and geometric computing. *Automata, Languages and Programming*, pages 7–16, 1997.

[40] H. Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996.

[41] G. Narasimhan. A note on the Hamiltonian circuit problem on directed path graphs. *Information Processing Letters*, 32(4):167–170, 1989.

[42] S. Rao Arikati and C. Pandu Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Information Processing Letters*, 35(3):149–153, 1990.

[43] G. Reinelt. TSPLIB–A traveling salesman problem library. *INFORMS Journal on Computing*, 3(4):376, 1991.

[44] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994.

[45] N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[46] G.C. Rota. The number of partitions of a set. *American Mathematical Monthly*, pages 498–504, 1964.

[47] I. Sau and D.M. Thilikos. Subexponential parameterized algorithms for degree-constrained subgraph problems on planar graphs. *Journal of Discrete Algorithms*, 8(3):330–338, 2010.

[48] P.D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[49] H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. *Algorithms-ESA 2003*, pages 765–775, 2003.

[50] R. Uehara. Simple geometrical intersection graphs. *WALCOM: Algorithms and Computation*, pages 25–33, 2008.