Graph Decomposition Based Algorithms for Maximum Path Coloring and Connected Subgraph Problems

by

Mehwish Bashir

M.Sc., (Computer Science), University of Agriculture (Faisalabad), 2007
M.Sc., University of Agriculture (Faisalabad), 2003

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

Spring 2012

APPROVAL

Name: Mehwish Bashir

Degree: Master of Science

Title of Thesis: Graph decomposition based algorithms for maximum path coloring and connected subgraph problems

Examining Committee: Dr. Arthur L. Liestman
Chair

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

Dr. Joseph G. Peters,
Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Jiangchuan Liu,
Associate Professor, Computing Science
Simon Fraser University
SFU Examiner

Date Approved: April 5, 2012

Partial Copyright Licence

Abstract

Many application problems in networks can be modeled as optimization problems in a graph G. For example, the maximum path coloring (Max-PC) problem, described next is an abstract model for many routing problems: Given a set P of paths in G and k colors, find a maximum subset of P and assign a color to each path of the subset such that the paths with the same color are edge-disjoint. One approach for solving an optimization problem in G is to decompose G into subgraphs, find partial solutions in each subgraph and combine the partial solutions into a solution of the problem. A carving-decomposition (branch-decomposition) of G is a system of edge-cuts (vertex-cuts) which decomposes G into subgraphs with each edge (vertex) a minimal subgraph. We give a carving-decomposition based exact algorithm and 1.58-approximation algorithm for the Max-PC problem. Let L be the maximum number of paths in P on any edge of G and let γ be the maximum cardinality of any edge-cut in a given carving-decomposition. Our exact algorithm and approximation algorithm run in $O((L+1)^{1.5k\gamma}n^2)$ and $O((L+1)^{1.5\gamma}kn^2)$ time, respectively. Our computational study shows that the exact algorithm can solve the Max-PC problem for small k and γ in a practical time and the approximation algorithm gives solutions close to optimal ones for practical values of k and L, and small γ. We also conduct a computational study on a branch-decomposition based exact algorithm for the maximum edge degree-bounded connected subgraph (MEDBCS) problem. Our result shows that the MEDBCS problem of vertex-degree bounded by 3 can be solved for graphs with small branchwidth in a practical time.

Keywords: Branch-decomposition, carving-decomposition, edge-disjoint paths, bidimensionality theory, grid minors

To my loving daughter, caring husband and my parents

"Success seems to be connected with action,
successful people keep moving, they make
mistakes, but they don't quit"

— Conrad Hilton

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my senior supervisor, Dr. Qian-Ping Gu for his continuous support, guidance during my research, for his motivation, and immense knowledge. His guidance helped me a lot in all the research work and particularly in writing of this thesis. I am indebted to him more than he knows.

Besides my senior supervisor, I would like to thank my supervisor, Dr. Joseph G. Peters, for insightful comments and suggestions. I am also grateful to the examiner, Dr. Jiangchuan Liu for reviewing my thesis and serving my defense committee. Many thanks to Dr. Arthur L. Liestman for his time to chair my defense. I would like to thank Zhengbing Bian, who lay a foundation of this research work. Many thanks go to Marjan Marzban for her help and guidance with the implementation of the algorithms.

I would like to thank my beloved parents for giving birth to me at the first place and supporting me physically and spiritually throughout my life. All that I am, or hope to be, its only because of their struggle, guidance and prayers. Last but not the least, I would like to thank my husband Majid Hussain, for his love, endless support and encouragement and to my little princess, Mahnoor Hussain.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graphs are well used models for computer and communication networks. A graph $G(V, E)$ consists of a set $V(G)$ of vertices and a set $E(G)$ of edges. A network can be represented by a graph G with vertices of G for network nodes and edges of G for network links. Many application problems in networks can be modeled as optimization problems in graphs, for example, some resource allocation problems as domination problems in graphs, routing problems in networks as disjoint path problems in graphs, and so on. A vast class of problems in graphs with wide and important applications, including those mentioned above, are NP-hard. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. One approach for solving an optimization problem in a graph G is to decompose G into subgraphs, find partial solutions of the problem in subgraphs and combine the partial solutions into a solution of the problem. Two methods for graph decompositions, branch-decompositions and carving-decompositions, have received much attention in the research of algorithms for NP-hard problems in graphs.

The notions of branchwidth/branch-decomposition are introduced by Robertson and Seymour [35]. The notions of carvingwidth/carving-decomposition are introduced by Seymour and Thomas in relation to branchwidth/branch-decomposition [39]. In this thesis, we denote by G an undirected graph and by $\vec{G}$ a directed graph. Informally, a *carving-decomposition* T_C of G is a system of edge-cut sets of G represented as edges of a tree with every leaf of the tree assigned a distinct vertex of G. The width of a carving-decomposition is the size of a maximum edge-cut in the decomposition. The *carvingwidth* $\mathrm{cw}(G)$ of G is the minimum width of all possible carving-decompositions of G. Informally, a *branch-decomposition* T_B of G is a system of vertex-cut sets of G represented as edges of a tree with

every leaf of the tree assigned a distinct edge of G. The width of a branch-decomposition is the size of a maximum vertex-cut in the decomposition. The *branchwidth* $\mathrm{bw}(G)$ of G is the minimum width of all possible branch-decompositions of G. Formal definitions of carving-decomposition, carvingwidth, branch-decomposition and branchwidth are given in Chapter 2.

The carving-decomposition/branch-decomposition based algorithms are designed to solve many optimization problems. The framework of these algorithms consists of two major steps. The first step is to compute a carving-decomposition/branch-decomposition of the given graph. The second step is to solve the problem by a dynamic programing approach, based on the decomposition computed in the first step. Usually the carving-decomposition/branch-decomposition based algorithms run in polynomial time in the size of G and exponential time in the width of the decomposition computed in the first step. Many NP-hard problems in G can be solved efficiently if the branchwidth or the carvingwidth of G is small. In the case of branch-decomposition based algorithms, if $\mathrm{bw}(G)$ is large, the theory of bidimensionality has been developed to deal with the problem.

A fundamental routing problem in computer and communication networks is that given a set of connection requests (source-destination pairs) in a network, find a path for each request and assign each path a channel such that the paths assigned the same channel do not share any communication link in the network. An important optimization goal for the routing problem is to accommodate as many requests as possible with a given number of channels. This optimization problem can be modeled as the maximum routing and path coloring (Max-RPC) problem in graphs: Given a set of source-destination vertex pairs in a graph G and k colors, find a path connecting a source-destination pair and assign the path one of the k colors for as many paths as possible such that the paths with the same color do not share any edge of G. When $k = 1$, the Max-RPC problem is known as the maximum edge-disjoint path (MEDP) problem. An important variant of the Max-RPC problem is the maximum path coloring (Max-PC) problem: Given a set P of paths in a graph G and k colors, select a maximum subset of P and assign each path in the subset a color such that the paths with the same color are edge-disjoint. When $k = 1$ the Max-PC problem is known as the maximum edge-disjoint paths with pre-specified paths (MEDPwPP) problem.

The Max-RPC and Max-PC problems have important applications in all-optical networks (and more generally circuit-switched networks) [13, 28, 31]. An optical network is

called all-optical if the optical signals from a source are transmitted to a destination without opto-electronic conversions at the intermediate node. Wavelength Division Multiplexing (WDM) is a widely used technology in optical networks to allow multiple requests to be carried on a same optical fiber by assigning a distinct wavelength to each request. The routing and wavelength assignment (RWA) problem is a fundamental problem in WDM optical networks: given k wavelengths and a set of connection requests in a network, find a path for each request and assign each path a wavelength such that the paths with the same wavelength do not share a communication link in the network. When the routing paths are given, the RWA problem becomes the wavelength assignment (WA) problem. The Max-RPC problem and Max-PC problem are mathematical models for the RWA problem and WA problem, respectively.

The Max-PC problem is a classical NP-hard problem. Given a set P of paths in a graph G, the *conflict graph* associated with P is the graph $G_c(P, E_c)$ with the vertex set P such that each vertex of G_c corresponds to a path in P and two vertices of G_c are adjacent if and only if the corresponding paths in P share an edge of G. The MEDPwPP problem (a special case of the Max-PC problem) in G is equivalent to the independent set problem in the conflict graph G_c. The independent set problem is NP-hard [18]. For any constant $\epsilon > 0$ it is not feasible to approximate the independent set problem in an arbitrary graph of n vertices within a factor of $n^{1-\epsilon}$ unless $P = NP$ [24]. Restricted to specific classes of graphs, a polynomial time exact algorithm is known for chains [7], for the MEDPwPP and Max-PC problems. Garg *et al.* in [20] propose an exact algorithm to solve the MEDP problem in undirected trees. Erlebach and Jansen design a 1.58-approximation algorithm for the Max-PC problem in undirected trees [14] by using the exact algorithm for the MEDP problem by Garg *et al.* [20]. Erlebach and Jansen prove that the MEDP problem is NP-hard in directed trees [16]. A $(5/3 + \epsilon)$-approximation algorithm is given in [16] for the MEDP problem in directed trees, where ϵ can be chosen arbitrarily small. The Max-PC problem in undirected and directed trees with $k > 1$ is studied in [13]. Erlebach use the iterative greedy approach to obtain 1.58-approximation algorithm for bounded degree directed trees and a 2.22-approximation algorithm for directed trees [13]. For path coloring problem, the best known algorithm for directed trees that colors a given set of paths with maximum load L using at most $(5/3)L$ colors [25, 26, 28, 31]. The Max-PC problem in undirected stars is NP-hard [15, 34]. For undirected and directed rings, the MEDPwPP problem can be solved in polynomial time, since the conflict graph is a circular-arc graph in this case,

and the maximum independent set problem is polynomial for circular-arc graphs [23]. For graphs as simple as rings, the Max-PC problem remains NP-hard [19]. For the Max-PC problem in rings, a 1.5-approximation algorithm is known [33]. Many heuristics including the first-fit, random-fit, most-used, and least-used have been developed for the Max-PC problem [2, 9, 32].

Little is known about exact algorithms for the Max-PC problem for arbitrary graphs. The Max-PC problem and a special case of the problem, the MEDPwPP problem, in planar graphs have been studied in [4]. More specifically, a carving-decomposition based exact algorithm for the MEDPwPP problem is developed in [4]. Given a set P of paths in a planar graph G of n vertices with L the maximum number of paths in P on any edge of G, the algorithm solves the MEDPwPP problem in $O((L+1)^{1.5\mathrm{cw}(G)}n^2 + n^3)$ time. Based on the MEDPwPP algorithm, a 1.58-approximation $O((L+1)^{1.5\mathrm{cw}(G)}kn^2 + n^3)$ time algorithm is given in [4] for the Max-PC problem in planar graphs. It is also mentioned in [4] that the MEDPwPP algorithm can be generalized to solve the Max-PC problem in planar graphs in $O((L+1)^{1.5k\mathrm{cw}(G)}n^2 + n^3)$ time. We extended the work of [4] by giving a carving-decomposition based exact algorithm for the Max-PC problem in arbitrary graphs. We also gave a 1.58-approximation algorithm for the Max-PC problem in arbitrary graphs. Our algorithms work not only for undirected graphs but also for directed graphs. We performed a computational study to evaluate the practical performances of our algorithms.

A *path* in G is a sequence $v_0e_1v_1...e_kv_k$, where $v_0, v_i \in V(G)$, $e_i = \{v_{i-1}, v_i\} \in E(G)$ for $1 \leq i \leq k$ and no vertex is repeated in the sequence. This sequence is called a *cycle* if $v_0 = v_k$. The length of the path (cycle) is the number of edges in the path (cycle). The *longest path (cycle) problem* is the problem of finding a path (cycle) of maximum length in a given graph. The decision version of the longest path (cycle) problem is NP-complete. A *Hamiltonian path* is a path in an undirected graph that visits each vertex exactly once. A *Hamiltonian cycle* is a cycle in an undirected graph that visits each vertex exactly once and also returns to the starting vertex. The longest path (cycle) problem is a generalization of Hamiltonian path (cycle) problem. In this thesis, we performed a computational study of the maximum edge degree-bounded connected subgraph (MEDBCS) problem, which is a generalization of the longest path problem and the Hamiltonian cycle problem.

A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The *maximum edge degree-bounded connected subgraph* (MEDBCS) problem is that: given a graph G, a positive integer $d \leq |V(G)|$, and a positive integer $k \leq |E(G)|$, decide if there is a connected

subgraph H of G such that $E(H) \geq k$ and every vertex of H has degree at most d. The optimization version of the MEDBCS problem is to find a largest connected subgraph H of G with vertex degree upper bounded by d. The longest path (cycle) problem is a special case of the MEDBCS problem with $d \leq 2$ ($d = 2$).

A graph parameter P is a function mapping graphs to positive integers. A graph H is a *minor* of G, if H is obtained by zero or more edge contractions on a subgraph of G. A $(r \times r)$-grid graph is a two-dimensional graph with r^2 vertices and edges between these vertices differ by ± 1 in exactly one coordinate. We denote by $gm(G)$ the largest integer r such that G contains a $(r \times r)$-grid as a minor. A problem is *bidimensional*, if the value of the parameter P depends on the $(r \times r)$-grid and $P(H) \leq P(G)$, where H is a minor of G. The aim is to decide if $P(G) \geq k$ for given G and k. In bidimensionality theory based algorithms, one first computes $\mathrm{bw}(G)$. If $\mathrm{bw}(G)$ is larger than some threshold value, then $P(G) \geq k$ may be concluded from $gm(G)$. Otherwise, a branch-decomposition based exact algorithm is used to solve the problem optimally by a dynamic programing approach. The MEDBCS problem is an example of bidimensional problems and one of the classical NP-hard problems listed in [18]. Recently it has been proved that it is not in APX for any fixed $d \geq 2$ [1]. The MEDBCS problem is a generalization of the longest path problem, where $d \leq 2$ and Hamiltonian cycle problem, where $d = 2$. Without the connectivity constraint, the problem can be solved in polynomial time using matching techniques [29]. Sau and Thilikos in [38] give the bidimensionality theory based subexponential parametrized algorithm for the MEDBCS problem. Sau and Thilikos in [38] use a sphere-cut decomposition of G and a non-crossing property of planar graphs. They show that the MEDBCS problem can be solved in $O(2^{log(5(d+1))6\sqrt{k}/\delta}\sqrt{k}n + n^3)$ time for planar graphs, where d is the degree and δ is a constant which depends on d. In this thesis, we performed a computational study on the bidimensionality theory based algorithm for the MEDBCS problem in planar graphs.

1.1 Contributions

1.1.1 Algorithms for the Max-PC problem and computational study

The Max-PC problem has important practical applications such as the routing in all-optical networks (and more generally circuit-switched networks) [13, 28, 31]. Recently, there have been increasing interests in developing exact algorithms for many NP-hard problems in

graphs. We give a carving-decomposition based exact algorithm, called Algorithm ALG-PC, for the Max-PC problem in G [3]. An input instance to Algorithm ALG-PC consists of a graph G, a set P of paths on G and k colors. We call a subset $Q \subseteq P$ a *feasible solution* if each path of Q can be assigned one of k colors and the paths with the same color are edge-disjoint. The algorithm outputs a maximum feasible solution $Q \subseteq P$. There are two major steps in Algorithm ALG-PC: (I) Compute a carving-decomposition T_C of small width for the graph G. (II) Use a dynamic programming approach based on T_C to compute a maximum feasible solution $Q \subseteq P$. Assume that Step (I) computes a carving-decomposition of width γ in $f(n)$ time, our algorithm solves the Max-PC problem in $O((L+1)^{1.5k\gamma}n^2 + f(n))$ time, where L is the maximum number of paths in P on any edge of G. When $k = 1$, the the Max-PC problem is reduced to the MEDPwPP problem. We also give a carving-decomposition based exact algorithm called Algorithm ALG-ED that solves the MEDPwPP problem in $O((L+1)^{1.5\gamma}n^2 + f(n))$ time. Based on the *iterative greedy approach* and Algorithm ALG-ED for the MEDPwPP problem, we give a 1.58-approximation $O((L+1)^{1.5\gamma}kn^2 + f(n))$ time algorithm for the Max-PC problem for $k > 1$: Find a maximum edge-disjoint subset of paths P by Algorithm ALG-ED and assign the paths in the subset one color; remove the subset from P and the assigned color; repeat the above process until no color is available or all paths of P have been colored.

It is NP-hard to compute an optimal carving-decomposition for arbitrary graphs [21]. For planar graphs, an optimal carving-decomposition (of width $\mathrm{cw}(G)$) can be computed in $O(n^3)$ time [21, 39]. By using this result, our exact algorithm solves the Max-PC problem in $O((L+1)^{1.5k\mathrm{cw}(G)}n^2+n^3)$ time and approximation algorithm runs in $O((L+1)^{1.5\mathrm{cw}(G)}kn^2+n^3)$ time for a planar graph G.

Many practical networks are modeled as directed graphs and the routing paths are directed. For example, an optical link between a pair of network nodes usually consists of a pair of directed optical fibers, one in each direction. Such a network is modeled as a directed graph with a pair of directed edges, one in each direction. Our algorithm works for directed graphs as well. Let $\vec{G}$ be a directed graph consisting of a set $V(\vec{G})$ of vertices and a set $E(\vec{G})$ of edges, where each edge is an ordered pair (u, v) of vertices. Edge (u, v) is called an edge from u to v. In the Max-PC problem on directed graphs, P is a set of directed paths. For a directed graph $\vec{G}$, the underline graph $H_{\vec{G}}$ of $\vec{G}$ is an undirected graph with $V(H_{\vec{G}}) = V(\vec{G})$ and $\{u, v\} \in E(H_{\vec{G}})$ if $(u, v) \in E(\vec{G})$ or $(v, u) \in E(\vec{G})$. For a directed graph $\vec{G}$, assume that Step (I) computes a carving-decomposition of $H_{\vec{G}}$ with width γ in

$f(n)$ time. Our exact algorithm runs in $O((L+1)^{3k\gamma}n^2 + f(n))$ time and the approximation algorithm runs in $O((L+1)^{3\gamma}kn^2 + f(n))$ time for the Max-PC problem in $\vec{G}$. For a simple directed planar graph $\vec{G}$, our exact algorithm runs in $O((L+1)^{3k\mathrm{cw}(H_{\vec{G}})}n^2 + n^3)$ time and the approximation algorithm runs in $O((L+1)^{3\mathrm{cw}(H_{\vec{G}})}kn^2 + n^3)$ time.

A ring is a well used topology in optical networks. An undirected ring is the graph $C(V, E)$ with $V(C) = \{u | 0 \leq u < n\}$ and $E(C) = \{\{u, v\} | u \equiv (v \pm 1) \bmod n\}$. It is easy to see that $\mathrm{cw}(C) = 2$ and a carving-decomposition of C with width 2 can be constructed in linear time. For undirected ring, our exact algorithm solves the Max-PC problem in $O((L+1)^{3k}n)$ time and the approximation algorithm runs in $O((L+1)^3kn)$ time. The directed ring is a well used topology in optical networks. A directed ring is the graph $\vec{C}(V, E)$ with $V(\vec{C}) = \{u | 0 \leq u < n\}$ and $E(\vec{C}) = \{(u, v), (v, u) | u \equiv (v \pm 1) \bmod n\}$. The directed ring $\vec{C}$ consists of two directed cycles, one in the clockwise direction and the other in the counter clockwise direction. The Max-PC problem on $\vec{C}$ can be solved on each of the cycles independently and the problem on each cycle can be viewed as the problem on the undirected ring. Therefore, our exact algorithm solves the Max-PC problem in $O((L+1)^{3k}n)$ time and the approximation algorithm runs in $O((L+1)^3kn)$ time for the directed rings.

We also conducted a computational study of our algorithms. We tested our algorithms on the graphs which are abstract models of a 16-node NSFNET (see Figures 3.2), a 24-node ARPANET (see Figure 3.3) and on the ring C. Our study shows that the practical performances of the algorithms coincide with the theoretical analysis, the exact algorithm is efficient for the instances with small k and L on graphs with small $\mathrm{cw}(G)$ (e.g, NSFNET and rings) but it is time consuming if one of k, L and $\mathrm{cw}(G)$ is large. An alternative approach is the approximation algorithm which computes near optimal solutions for the Max-PC problem efficiently even for large k and L with small $\mathrm{cw}(G)$.

We also implemented the first-fit, random-fit, most-used and least-used heuristics [2, 9, 32]. For the NSFNET, we compared the results of our 1.58 approximation algorithm with the results of heuristics. It is observed that, in the NSFNET, for the Max-PC problem our approximation algorithm colors more paths than the heuristics. We also implemented the 1.5-approximation algorithm (NPZ algorithm) [33] for the Max-PC problem on rings. For the ring, we compared the results of our 1.58-approximation algorithm with the results of 1.5-approximation algorithm (NPZ algorithm) and the heuristics. We showed that the 1.58-approximation algorithm is also efficient for the the Max-PC problem on the ring with practical values of k and L. The NPZ algorithm achieved the best known approximation

ratio (1.5) for the Max-PC problem on the ring. The results show that, for the ring, our 1.58-approximation algorithm has a similar performance in the number of paths colored as the NPZ algorithm and colors more paths than the first-fit, random-fit, most-used and least-used heuristics. On the other hand, the 1.58-approximation algorithm works for general graphs and has a comparable performance as that of NPZ algorithm on the ring.

1.1.2 Computational study for the MEDBCS problem

The longest path problem and Hamiltonian cycle problem are NP-hard. In [12] Dorn et al. give a branch-decomposition based algorithm which solves Hamiltonian path like problems in planar graphs in $2^{O(\mathrm{bw}(G))}n^{O(1)}$ time. Based on the sphere-cut decomposition (sc-decomposition), which is a special type of branch-decomposition, and the non-crossing property of planar graphs embedded on a sphere, 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 [12]. They also show that planar Hamiltonian cycle problem can be solved in $O(2^{6.903\sqrt{n}})$ time.

We performed a computational study on the bidimensionality theory based algorithm for the MEDBCS problem [38] which is a generalization of the longest path problem, where $d \leq 2$ or the Hamiltonian cycle problem, where $d = 2$. We implemented the branch-decomposition based algorithm for solving the MEDBCS problem. In the first step we compute sphere-cut decomposition (sc-decomposition). It is known that an optimal sc-decomposition of a planar graph embedded on a sphere, can be computed in $O(n^3)$ [21, 39]. We used the tools for computing $\mathrm{bw}(G)$ and optimal branch-decomposition of G, reported in [5, 6].

After finding $\mathrm{bw}(G)$, there are two cases: If $\mathrm{bw}(G) \leq 3\sqrt{k}/\delta$, where δ is a constant depending on the bounded degree d: for $d = 2$ $\delta = 1$, for $d = 3$ $\delta = \sqrt{3/2}$ and for $d \geq 4$ $\delta = \sqrt{2}$, the value of the MEDBCS is computed in $O(2^{log(5(d+1))6\sqrt{k}/\delta}\sqrt{k}n)$ time by using the dynamic programing algorithm [38] . Otherwise, the value of the MEDBCS problem is computed from the largest grid minor. For computing the largest grid minors, we use the tool reported in [41] called GT tool. The GT tool is an implementation of Gu and Tamaki's algorithm, proposed in [22]. The GT tool finds a $(g \times h)$-cylinder minor, where $g \geq 3$ and $h \geq 1$. Notice that a $(g \times h)$-cylinder contains a $(g \times h)$-grid as a minor. It is shown in [22] that for a planar graph G, a $(g \times g/2)$-cylinder minor with $g \geq bw(G)/2$, can be computed efficiently. From this, we could conclude that for the MEDBCS problem $P(G) \geq c(\mathrm{bw}(G))$ for some constant $c > 0$.

1.2 Thesis outline

The rest of the thesis is organized as follows. In Chapter 2, we give the preliminaries of the thesis. In Chapter 3, algorithms and a computational study for the Max-PC problem are introduced. We present a computational study for bidimensionality theory based algorithm for the maximum edge bounded-degree connected subgraph (MEDBCS) problem in Chapter 4. The final chapter concludes the thesis.

Chapter 2

Preliminaries

We denote by G an undirected simple graph which consists of a set $V(G)$ of vertices and a set $E(G)$ of edges, where each edge e of $E(G)$ is a subset of $V(G)$ with two elements. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a subset $U \subseteq V(G)$ ($E' \subseteq E(G)$), we denote by $G[U]$ ($G[E']$) the subgraph of G induced by $U(E')$.

The notions of carvingwidth and carving-decomposition are introduced by Seymour and Thomas in relation to branchwidth and branch-decomposition [39]. A *carving-decomposition* of G is a tree T_C where each internal node of T_C has degree 3 and each leaf node of T_C is associated with a distinct vertex of G. Removing a link e of T_C separates T_C into two subtrees T_1 and T_2. Let V' and V'' be the sets of leaves of the two subtrees. The *middle set* denoted by $E_{mid(e)}$ is the set of edges with an end vertex in V' and an end vertex in V''. The width of the link e is $|E_{mid(e)}|$. We define the width of the carving-decomposition T_C to be the maximum width of all links of T_C. The *carvingwidth* of G, denoted by $\mathrm{cw}(G)$, is the minimum width of all carving-decompositions of G. Figure 2.1 gives an example of a carving-decomposition of a graph G.

The notions of branchwidth and branch-decomposition were introduced by Robertson and Seymour [35]. A *branch-decomposition* of G is a tree T_B where each internal node of T_B has degree 3 and set of leaves of T_B is associated with the edge set of G. For every link e of T_B, let T_1 and T_2 be the two connected components obtained after removing a link e from T_B. Let G_1 and G_2 be the subgraphs induced by the edge sets of T_1 and T_2, respectively. The *middle set* denoted by $V_{mid(e)}$ is the intersection of the vertex sets of G_1 and G_2. The width of the link e is $|V_{mid(e)}|$. We define the width of the branch-decomposition T_B to be the maximum width of all links of T_B. The *branchwidth* of G, denoted by $\mathrm{bw}(G)$, is

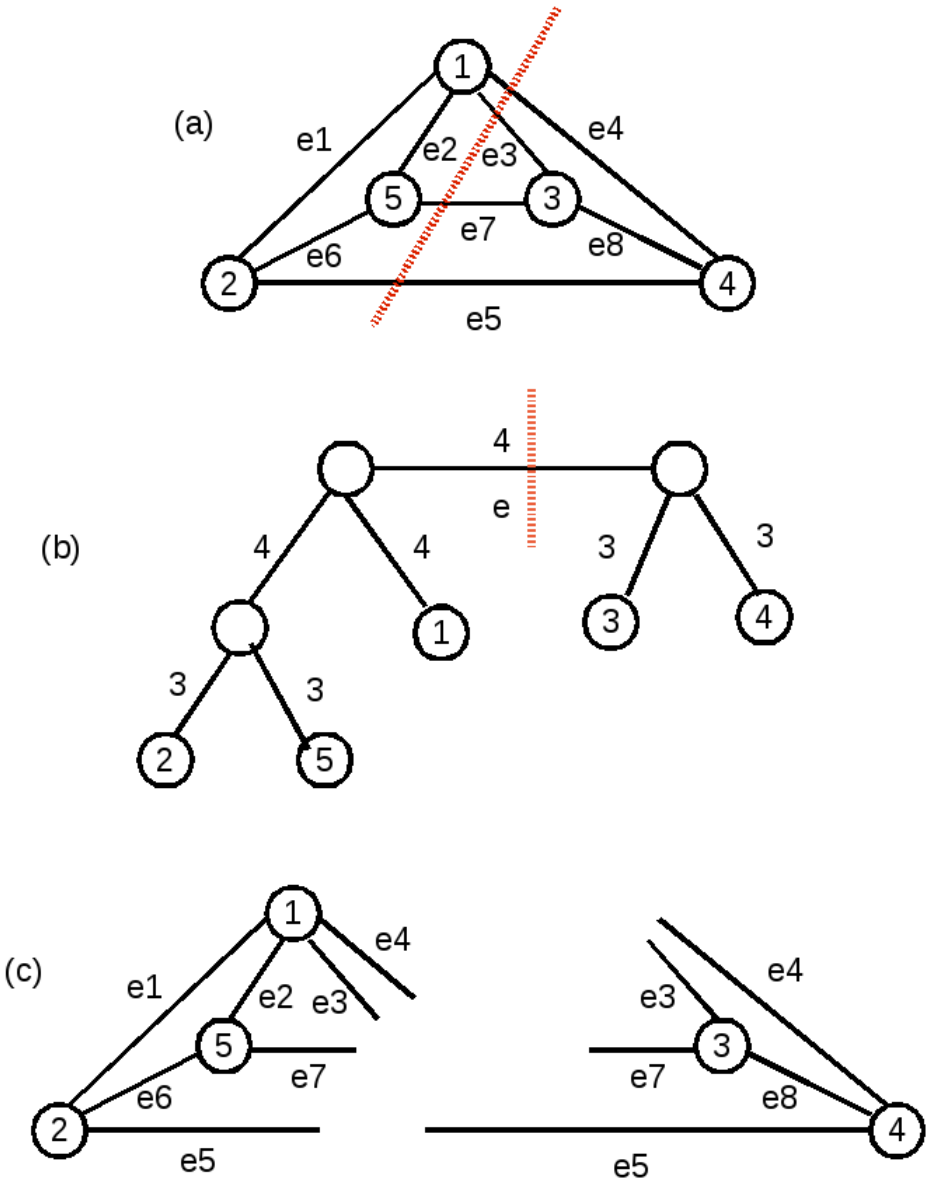

Figure 2.1: (a) A graph G, (b) a carving-decomposition T_C of G. The number on each link e of T_C is $|E_{mid(e)}|$ for that link. T_C has width 4 and (c) two subgraphs of G induced by V' and V''. The edge-cut induced by link e of T_C is $E_{mid(e)} = \{e_3, e_4, e_5, e_7\}$.

the minimum width of all branch-decompositions of G. Figure 2.2 gives an example of a branch-decomposition of a graph G.

A carving-decomposition T_C (resp. branch-decomposition T_B) of G can be converted to a binary tree with root r by replacing an internal link $e = \{x, y\}$ with three links $\{x, z\}, \{z, y\}, \{r, z\}$, where z and r are new nodes to T_C (resp. T_B), r is the root, and $\{z, r\}$ is an internal link. For every internal link e of T_C (resp. T_B), e has two children links incident to e. For every link e of T_C (resp. T_B), let T_e be the subtree of T_C (resp. T_B) consisting of all descendant links of e. Let H_e be the subgraph of G induced by the vertices (resp. edges) at leave nodes of T_e.

We denote by $\vec{G}$ a *directed graph* consisting of a set $V(\vec{G})$ of vertices and a set $E(\vec{G})$ of edges, where each edge is an ordered pair (u, v) of vertices. Edge (u, v) is called an edge from u to v. For a directed simple graph $\vec{G}$, we define the *underline graph* $H_{\vec{G}}$ of $\vec{G}$ to be an undirected graph with $V(H_{\vec{G}}) = V(\vec{G})$ and $\{u, v\} \in E(H_{\vec{G}})$ if $(u, v) \in E(\vec{G})$ or $(v, u) \in E(\vec{G})$.

A *path* in G (resp. $\vec{G}$) is a sequence $v_0 e_1 v_1 ... e_k v_k$, where $v_0, v_i \in V(G)$ (resp. $v_0, v_i \in V(\vec{G})$), $e_i = \{v_{i-1}, v_i\} \in E(G)$ (resp. $e_i = (v_{i-1}, v_i) \in E(\vec{G})$) for $1 \leq i \leq k$ and no vertex is repeated in the sequence. This sequence is called a *cycle* if $v_0 = v_k$. The length of the path (cycle) is the number of edges in the path (cycle). The *distance* between two vertices in a graph is the length of the shortest path between them. We say a path is on an edge, if the edge appears in the sequence of the path. Given a set A of edges and a set P of paths in a graph, we say P is on A, if every path of P is on some edges of A. Given a set P of paths in a graph and an edge e of the graph, we denote by $L(e)$ the number of paths in P that are on e. We call $L(e)$ the load of P on e. The *load* of the graph, denoted by L, is the maximum $L(e)$ of any edge e in the graph. The *degree* of a vertex v of G denoted by $deg(v)$, is the number of edges incident to v.

An *undirected ring* is the graph $C(V, E)$ with $V(C) = \{u | 0 \leq u < n\}$ and $E(C) = \{\{u, v\} | u \equiv (v \pm 1) \bmod n\}$. It is easy to see that $\mathrm{cw}(C) = 2$ and a carving-decomposition of C with width 2 can be constructed in linear time. A *directed ring* is the graph $\vec{C}(V, E)$ with $V(\vec{C}) = \{u | 0 \leq u < n\}$ and $E(\vec{C}) = \{(u, v), (v, u) | u \equiv (v \pm 1) \bmod n\}$. The directed ring is a well used topology in optical networks. The directed ring $\vec{C}$ consists of two directed cycles, one in the clockwise direction and the other in the counter clockwise direction. A graph is *planar* if it can be drawn on a sphere without crossing edges.

Given an edge e of G which is between u and v, where $\{u, v\} \in V(G)$, the *edge contraction*

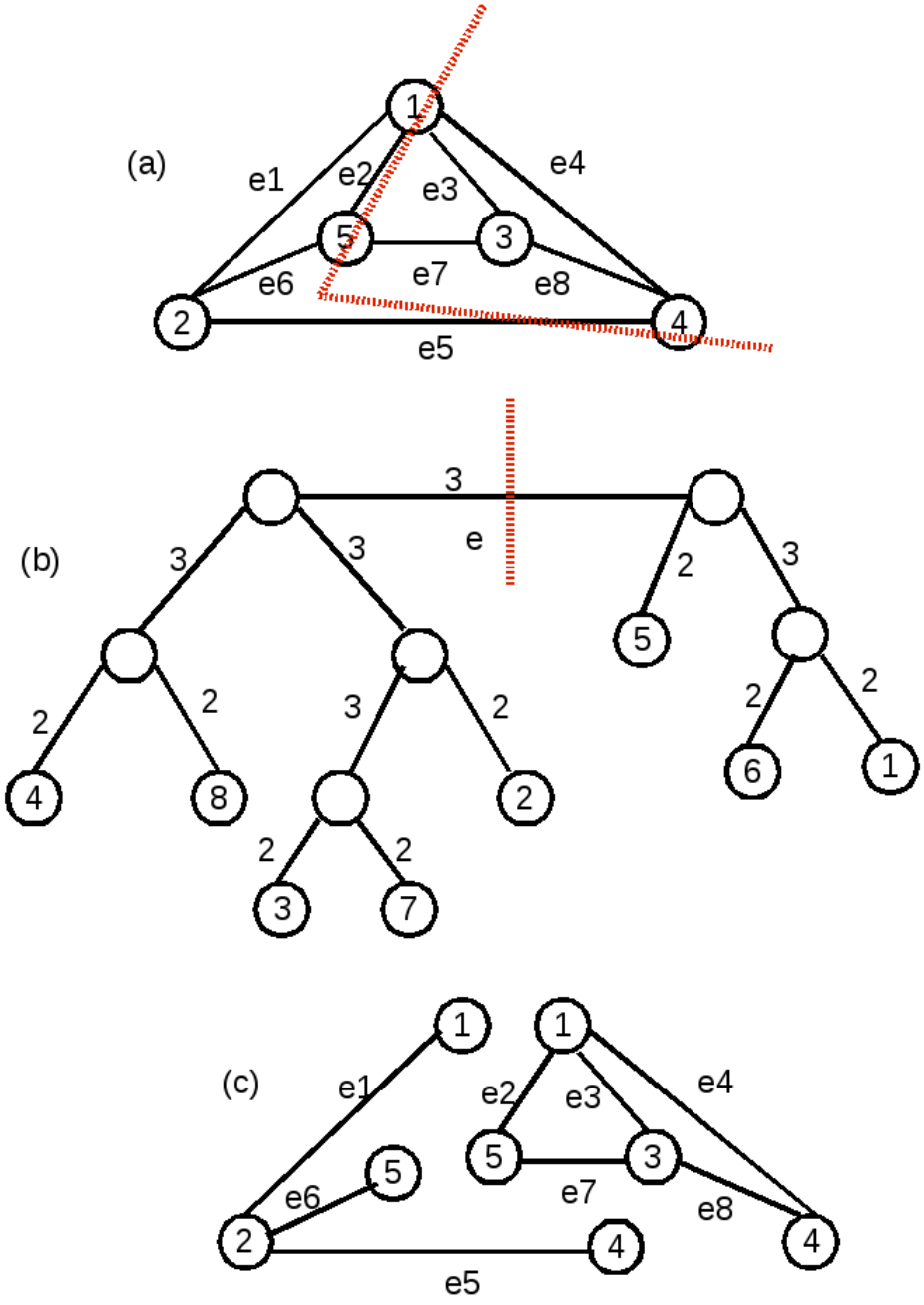

Figure 2.2: (a) A graph G, (b) a branch-decomposition T_B of G. The number on each link e of T_B is $|V_{mid(e)}|$ for that link. T_B has width 3 and (c) two subgraphs G_1 and G_2 of G. The vertex-cut induced by link e of T_B is $V_{mid(e)} = \{v_1, v_4, v_5\}$.

is to remove the edge e from G and u, v are merged into a new vertex w, where $w \notin V(G)$, whose incident edges are the edges that were incident to u or v other than e. A graph H which is obtained by a sequence of zero or more edge contractions is said to be a *contraction* of G. A graph H is a *minor* of G, if H is obtained by zero or more edge contractions of a subgraph of G. Checking whether H is a minor of G is solvable in $O(n^3)$ [36]. An $(r \times r)$-*grid* is a planar graph with r^2 vertices $\{(a, b)|1 \leq a, b \leq r\}$ with edges between vertices differing by ± 1 in exactly one coordinate. The size of the largest grid minor of G, denoted by $gm(G)$, is the largest integer r such that G contains a $(r \times r)$- grid as a minor. 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 $|j - j'| = 1$. Notice that a $(g \times h)$-cylinder contains a $(g \times h)$-grid as a minor. Let $cm(G)$ denote the largest integer g such that G contains a $(g \times g/2)$-cylinder as a minor. Gu and Tamaki show that for a planar graph G, $bw(G) \leq 2cm(G)$ and they also design an algorithm that computes a $(g \times h)$-cylinder minor of G [22].

An *isomorphism* from a graph G to a graph H is a bijection $f : V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ if and only if $\{f(u), f(v)\} \in E(H)$. A *graph property* is defined to be a property preserved under all possible isomorphisms of a graph. In other words, it is a property of the graph itself, not of a specific drawing or representation of the graph. A graph property is *minor-closed* if for each graph with a specific property, it holds that all its minors also have that specific property. For example, planarity of a graph is minor closed.

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 $\{(x, 0)|0 < x < 1\}$ in the sphere. For a topological segment S, the closure of S is denoted by $\overline{S}$ and $bd(S) = \overline{S} \setminus S$. We call the two elements of $bd(S)$ the end points of S. A *planar embedding* of a graph G is a mapping ϕ from $V(G) \cup E(G)$ to $\Sigma \cup 2^{\Sigma}$, satisfying the following properties:

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

If a graph has planar embedding then the graph is called *planar*. The planar embedding (G, ϕ) of graph G is called a *plane* graph. In what follows, we also denote by G a plane

graph (G, ϕ), leaving ϕ implicit. A *region* of a plane graph is a connected component of $\Sigma \setminus (E(G) \cup V(G))$. Let G be a plane graph. A *curve* in Σ is the image of a continuous function $f : [0, 1] \to \Sigma$. A curve is *G-normal* if it does not intersect with itself and meets G only at vertices of G. A curve is *closed* if $f(0) = f(1)$. A closed G-normal curve is a *noose*. The length of the noose is the number of vertices it meets. Let O be a noose of G that separates G into two regions R_1 and R_2. Then O induces a separation $(A, \overline{A})$ of G where $A = \{e \in E(G) | \phi(e) \subseteq R_1\}$ and $\overline{A} = \{e \in E(G) | \phi(e) \subseteq R_2\}$. A separation is called *noose induced*, if it is induced by some noose.

A branch-decomposition T_B is a *sphere-cut decomposition* or *sc-decomposition*, if every separation induced by a link of T_B is noose induced [12]. Formally, a sphere-cut decomposition T_B, is a special kind of branch decomposition, for every edge e of T_B there exists a noose O_e bounding the two open disks Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup O_e, 1 \leq i \leq 2$. Thus O_e meets G only in $V_{mid(e)}$ and its length is $|V_{mid(e)}|$. A clockwise traversal of O_e in the drawing of G defines the cyclic ordering π of $V_{mid(e)}$ and the vertices of every middle set are enumerated according to π. A sc-decomposition can be computed in time $O(n^3)$ [21, 39].

The *maximum routing and path coloring* (Max-RPC) problem in graphs is: Given a set of source-destination vertex pairs in a graph G and k colors, find a path connecting a source-destination pair and assign the path one of the k colors for as many paths as possible such that the paths with the same color do not share any edge of G. When $k = 1$, the Max-RPC problem is known as the *maximum edge-disjoint path* (MEDP) problem. An important variant of the Max-RPC problem is the *maximum path coloring* (Max-PC) problem: Given a set P of paths in a graph G and k colors, select a maximum subset of P and assign each path in the subset a color such that the paths with the same color are edge-disjoint. When $k = 1$ the Max-PC problem is known as the *maximum edge-disjoint paths with pre-specified paths* (MEDPwPP) problem.

The *maximum edge degree-bounded connected connected subgraph* (MEDBCS) problem is that: given a graph G, a positive integer $d \leq |V(G)|$, and a positive integer $k \leq |E(G)|$, decide if there is a connected subgraph H of G such that $E(H) \geq k$ and every vertex of H has degree at most d. The optimization version of the MEDBCS problem is to find a largest connected subgraph H of G with vertex degree upper bounded by d. The longest path (cycle) problem is a special case of the MEDBCS problem with $d \leq 2$ ($d = 2$).

A *parameter* P is a function mapping graphs to positive integers. For example the number of vertices of a graph and the number of edges of the graph. A problem is *bidimensional*

if the value of the parameter P depends on the size of the grid and $P(H) \leq P(G)$, where H is minor of G. The aim is to decide if $P(G) \leq k$ for given G and k. Bidimensionality is defined by Demaine *et al.* in [11] as follows: A parameter P is *minor bidimensional* with density δ if

1. P is closed under taking minors
2. for the $(r \times r)$-grid R, $P(R) = (\delta r)^2 + o((\delta r)^2)$

A parameter P is *contraction bidimensional* with density δ if

1. P is closed under contractions
2. for any partially triangulated $(r \times r)$-grid R, $P(R) = (\delta_R r)^2 + o((\delta_R r)^2)$
3. δ is the smallest δ_R among all partially triangulated $(r \times r)$-grid

The parameter P is called *bidimensional* either it is minor bidimensional or contraction bidimensional. The parameter is called *h(r)-bidimensional*, if it is at least $h(r)$ in a grid, where $h(r)$ is a function that depends on $(r \times r)$-grid. A parameter P is *minor-closed* (resp. *contraction-closed*) if for every graph H, which is a minor (resp. a contraction) of G, $P(H) \leq P(G)$. The density is usually $0 < \delta \leq 1$. For example a vertex cover problem. A parameter vertex cover is minor-bidimensional and for a $(r \times r)$-grid, the size of vertex cover is at least $r^2/2$. Therefore, a parameter vertex cover has density $1/\sqrt{2}$.

A parametrized problem is *fixed parameter tractable* (FPT) with respect to k if there exists an algorithm that computes the solution in $f(k).n^{O(1)}$ time, where n is the size of the graph and f is a computable function of k which is independent of n [17]. For example a vertex cover of size k can be found in $O(1.2745^k k^4 + kn)$ time [8]. If $f(k)$ is subexponential in k, may be $f(k) = 2^{O(\sqrt{k})}$, then the algorithm is called a *subexponential parametrized algorithm* that solves the parametrized problem in $2^{O(\sqrt{k})}.n^{O(1)}$ time.

In the bidimensionality theory based algorithms, the relationship between the branch-width and the size of the largest grid minor $gm(G)$ is important. Robertson, Seymour and Thomas show that $gm(G) \leq \text{bw}(G) \leq 20^{2gm(G)(gm(G)+1)^4}$ [37] for arbitrary graphs and $gm(G) \leq \text{bw}(G) \leq 4gm(G)$ for planar graphs [37]. For planar graphs, Gu and Tamaki improve the result to $\text{bw}(G) \leq 3gm(G)$ for a $(r \times r)$-grid minor [22]. For a $(g \times h)$-cylinder minor, $\text{bw}(G) \leq 2cm(G)$ [22], where $cm(G)$ is size of the largest cylinder minor. In the bidimensionality theory based algorithms, first computes $\text{bw}(G)$. If $\text{bw}(G)$ is larger than

some threshold value, then $P(G) \leq k$ may be concluded from $gm(G)$ or $cm(G)$. Otherwise, $P(G)$ is computed optimally by using the dynamic programing approach.

Chapter 3

Algorithms for the Max-PC Problem

In this Chapter, we describe algorithms for the MEDPwPP problem and Max-PC problem. To explain the steps of both of the algorithms, we need some definitions.

A network is modeled as an undirected graph G. A request in G is given by a path with a source node and destination node. A set of requests in a network is represented by a set of pre-specified paths P in a graph G. Given a path $p \in P$ in the graph G, we say p is on an edge $e \in E(G)$, if p contains e. Given a set of edges $E' \subseteq E(G)$ and a set of paths $P' \subseteq P$, we say P' is on E', if every path of P' contains an edge of E'. Given a set P of paths in G, for each edge e of G let $L_P(e) = |\{p|p \in P \text{ and } p \text{ is on } e\}|$, $L_P = \max_{e \in E(G)} L_P(e)$. In the rest of the thesis, $L(e)$ will be used for $L_P(e)$ and L for L_P, where L is called the load of the graph.

3.1 Algorithm for the edge-disjoint path problem

A special case of the Max-PC problem is the MEDPwPP problem, where $k = 1$. In this section, we first describe a carving-decomposition based exact algorithm called *Algorithm ALG-ED*, for the MEDPwPP problem.

An input instance to Algorithm ALG-ED consists of a graph G, a set P of paths on G. We call a subset $P' \subseteq P$ a *feasible solution* if all paths of P' are edge-disjoint. The algorithm outputs a maximum feasible solution $P' \subseteq P$. There are two major steps in

Algorithm ALG-*ED*: (I) Compute a carving-decomposition T_C of small width for the graph G. (II) Use a dynamic programming approach based on T_C to compute a maximum feasible solution $P' \subseteq P$. Step (II) has exponential time in the width of T_C computed in Step (I). So finding a carving-decomposition of small width is critical in reducing the running time of the algorithm. For a planar graph G, an optimal carving-decomposition can be computed in $O(n^3)$ time [21, 39]. Notice that for any graph G and a subgraph G' of G with $V(G') = V(G)$, a carving-decomposition of G' is also a carving-decomposition of G. From this fact, for a non-planar G, we can find a planar subgraph G' of G with $V(G') = V(G)$ and compute an optimal carving-decomposition T' of G' as a carving-decomposition of G. For small graphs, the planar subgraphs may be found by hand. However, for large graphs, one may rely on some heuristics to find planar subgraphs. Since G has more edges than G', T' may not be an optimal carving-decomposition of G. However, T' is very close to optimal if G is close to planar (this often happens in practice). One may use other heuristics in Step (I) for non-planar graphs.

In Step (II), the carving decomposition T_C is first converted to a rooted binary tree by replacing an internal link $e = \{x, y\}$ with three $\{x, z\}$, $\{z, y\}$, $\{r, z\}$ links. In these links z and r are new nodes added in T_C, where r is the root and $\{z, r\}$ is an internal link. For every internal link e of T_C, e has two child links incident to e. Let T_e be the subtree of T_C consisting of all the descendant links of e and H_e be the subgraph of G induced by the vertices at the leaf nodes of T_e. The dynamic programming step finds the partial solutions of H_e for every subtree T_e of T_C from leaves to the root in a bottom-up way. The partial solutions of H_e for each leaf link e is empty and the solutions for an internal link e is computed by merging the partial solutions for the child links of e, that are already computed.

For an internal link e of T_C, we use $\mathcal{P}_e$ to denote the set of all subsets of edge disjoint paths in P on $E_{mid(e)}$. For a set of edge disjoint paths $P'_e \in \mathcal{P}_e$, we define $f(e, P'_e)$ as $|Q_e|$, where Q_e is a maximum subset of paths in H_e such that $P'_e \cup Q_e$ is edge disjoint. Initially $f(e, P'_e)$ is set to 0 for all links e of T_C and for every possible subset $P'_e \in P_e$. For a leaf link, no computation is needed. An internal link e of T_C has two child links f and g. Let $X_1 = E_{mid(e)} \cap E_{mid(f)}$, $X_2 = E_{mid(e)} \cap E_{mid(g)}$ and $X_3 = E_{mid(f)} \cap E_{mid(g)}$. Then $X_1 \cup X_2 = E_{mid(e)}$, $X_1 \cup X_3 = E_{mid(f)}$ and $X_2 \cup X_3 = E_{mid(g)}$.

For an internal link e, we have two dynamic programing tables corresponding to the child links f and g. For each edge h of $E_{mid(f)}$ and $E_{mid(g)}$, every path on h is given a unique

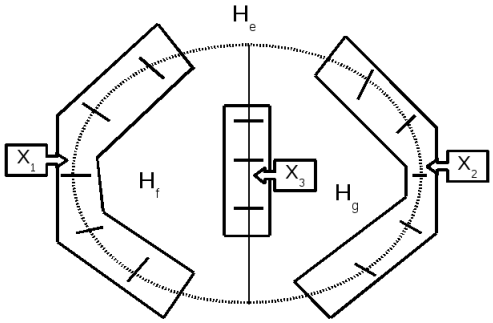

Figure 3.1: Subsets of cut sets $E_{mid(e)}, E_{mid(f)}, E_{mid(g)}$.

label from $\{1, 2, 3, ..., L\}$, where L is the maximum number of paths of P on any edge of G. We assume that $E_{mid(f)} = \{e_1, e_2, ..., e_{|E_{mid(f)}|}\}$ and $E_{mid(g)} = \{e_1, e_2, ..., e_{|E_{mid(g)}|}\}$. We use $\mathcal{P}_f$ and $\mathcal{P}_g$ to denote the set of all subsets of edge disjoint paths in P on $E_{mid(f)}$ and $E_{mid(g)}$, respectively. A set of edge disjoint paths $P'_f \in \mathcal{P}_f$ is represented by $\lambda_f(h) = \{l_1, l_2, ..., l_{|E_{mid(f)}|}\}$ where $l_i \in \{0, 1, ..., L\}$ with $l_i = 0$ denoting that no path on e_i appears in P'_f and $l_i = j \in \{1, ..., L\}$ denoting the path on e_i is assigned a label j appears in P'_f. Similarly, a set of edge disjoint paths $P'_g \in \mathcal{P}_g$ is represented by $\lambda_g(h) = \{l_1, l_2, ..., l_{|E_{mid(g)}|}\}$ where $l_i \in \{0, 1, ..., L\}$ with $l_i = 0$ denoting that no path on e_i appears in P'_g and $l_i = j \in \{1, ..., L\}$ denoting the path on e_i is assigned a label j appears in P'_g. While merging the solutions, again we have a table for the internal link e with the combinations of labels of the paths. We say a label λ_e is formed from a label λ_f and a label λ_g if

- for $h \in X_1$, $\lambda_e(h) = \lambda_f(h)$,
- for $h \in X_2$, $\lambda_e(h) = \lambda_g(h)$, and
- for $h \in X_3$, $\lambda_f(h) = \lambda_g(h)$.

It is important to mention here that while merging the solutions, reordering of the edges according to recently calculated X_1,X_2 and X_3 might be needed.

As mentioned earlier, we use $\mathcal{P}_e$ to denote the set of all subsets of edge disjoint paths in P on $E_{mid(e)}$. For every $P'_e \in \mathcal{P}_e$, $f(e, P'_e)$ is computed from $f(f, P'_f)$ and $f(g, P'_g)$. Let $\mathcal{P}''$ be a set of all possible subsets of edge disjoint paths from the corresponding set of subsets

P'_f and set of subsets P'_g and these paths are on the edges of $E_{mid(f)} \cup E_{mid(g)}$. For every $P'' \in \mathcal{P}''$, let $P''_e \subseteq P''$, $P''_f \subseteq P''$ and $P''_g \subseteq P''$ are sets of paths on some edges in $E_{mid(e)}$, $E_{mid(f)}$ and $E_{mid(g)}$, respectively. We initialize $f(e, P''_e)$ to 0. For every P'', first we compute $|(P''_f \cup P''_g) \setminus P''_e|$. Then the values $|(P''_f \cup P''_g) \setminus P''_e|$, $f(f, P''_f)$ and $f(g, P''_g)$ are added up. If this value is greater than the previous value of $f(e, P''_e)$, then $f(e, P''_e)$ is updated to this value. Thus, $f(e, P'_e)$ takes the maximum over all $f(e, P''_e)$. If both f and g are the leaves of T_C then $f(f, P''_f)$ and $f(g, P''_g)$ are 0. At the root link of T_C the maximum value of $f(e, P'_e)$ over all P'_e is the solution for a maximum edge disjoint path problem.

In order to calculate the running time of Algorithm ALG-ED, for Step (I) assume that Algorithm ALG-ED computes a carving decomposition T_C of G with width γ in $f(n)$ time, and Step (II) plays a major role in the time complexity. For each internal link e of T_C, there are $(L+1)^{|X_1|+|X_2|}$ or $(L+1)^{|E_{mid(e)}|}$ possible subsets of partial solutions to store. Since $E_{mid(e)}$ is bounded by the carvingwidth of the graph G γ, each edge of $E_{mid(e)}$ has a load bounded by L and each subset of partial solution contains at most one path on each edge of $E_{mid(e)}$. Therefore, there are at most $(L+1)^{\gamma}$ possible subsets of partial solutions to store. While merging, we only need to consider the paths on X_1, X_2 and the paths on X_3. Since $|X_1 \cup X_2 \cup X_3| \leq (1.5\gamma)$ there are at most $(L+1)^{1.5\gamma}$ cases to consider. The time complexity to process one link would be $O((L+1)^{1.5\gamma}n)$ and the memory requirement is $O((L+1)^{\gamma})$. Total time complexity and memory requirement would be $O(((L+1)^{1.5\gamma}n^2) + n^3)$ and $O((L+1)^{\gamma}n)$ respectively, where n represents number of vertices in the graph G.

For planar graph G, the carvingwidth of the graph G, cw(G) can be computed in O(n^3) time [21, 39]. Therefore, the time complexity and memory requirement for planar graphs would be $O(((L+1)^{1.5\mathrm{cw}(G)}n^2) + n^3)$ and $O((L+1)^{\mathrm{cw}(G)}n)$ respectively, where n represents number of vertices in the planar graph G. The running time and memory requirement are polynomial in the input parameters, when cw(G) is bounded by a constant.

3.2 Exact algorithm for the Max-PC problem

The algorithm for the MEDPwPP problem described in the previous section can be generalized to an exact algorithm for the Max-PC problem, called *Algorithm ALG-PC*. An input instance to Algorithm ALG-PC consists of a graph G, a set P of paths on G and k colors. We call a subset $Q \subseteq P$ a *feasible solution* if each path of Q can be assigned a color and the paths with the same color are edge-disjoint. The algorithm outputs a maximum

feasible solution $Q \subseteq P$. There are two major steps in Algorithm ALG-PC: (I) Compute a carving-decomposition T_C of small width for the graph G. (II) Use a dynamic programming approach based on T_C to compute a maximum feasible solution $Q \subseteq P$.

To solve the Max-PC problem or an instance of the Max-PC problem, first the carving decomposition T_C, which is computed in Step (I), is converted into a rooted binary tree, as done in the case of the exact algorithm for the MEDPwPP problem in section 3.1. The dynamic programming step finds the partial solutions of H_e for every subtree T_e of T_C from leaves to the root in a bottom-up way.

The following observation is useful for understanding the dynamic programming step.

Observation 3.2.1 *For a maximum feasible solution $Q \subseteq P$ and a link e in a carving-decomposition T_C of G, let $Q_e = Q'_e \cup Q''_e$ be the subset of Q such that each path of Q_e is on an edge of $E_{mid(e)} \cup E(H_e)$, where Q'_e is the set of paths on an edge of $E_{mid(e)}$ and Q''_e is the set of paths on an edge of H_e but not on any edge of $E_{mid(e)}$. Then*

1. *each edge of $E_{mid(e)}$ appears in at most k paths of Q'_e and*
2. *Q''_e is a maximum subset of P such that each path of Q''_e is on an edge of H_e but not on any edge of $E_{mid(e)}$, and $Q'_e \cup Q''_e$ is a feasible solution.*

We call a subset Q_e of P satisfying (1) and (2) in Observation 3.2.1 a *candidate* w.r.t. $E_{mid(e)}$. Notice that for a fixed Q'_e, there may be multiple sets of Q''_e satisfying (2). However, any such a Q''_e can be used to form a candidate Q_e w.r.t. $E_{mid(e)}$ because a path of Q''_e does not intersect with any path which is not on an edge of H_e and only the cardinality of Q''_e affects the size of a final solution. Therefore, the candidates w.r.t. $E_{mid(e)}$ can be identified by the sets Q'_e satisfying (1). We further identify every set Q'_e satisfying (1) by assigning labels to the edges of $E_{mid(e)}$: For each edge h of $E_{mid(e)}$, we give every path on h a unique index w.r.t. h from $\{1, 2, ..., L\}$, where L is the maximum number of paths of P on any edge of G. Since there are at most L paths on any edge of $E_{mid(e)}$, the indexing above can be done. There are $\sum_{i=0}^{k} \binom{L}{i}$ subsets of $\{1, 2, ..., L\}$. For coloring schemes, select only those subsets having cardinality at most i. For each subset with cardinality i, there are $\binom{k}{i}$ $i!$ coloring schemes. Since there are $\sum_{i=0}^{k} \binom{L}{i}$ subsets of $\{1, 2, ..., L\}$, the total number of coloring schemes for paths on one edge of $E_{mid(e)}$ is

$$\begin{aligned}\sum_{i=0}^{k}\binom{L}{i}\binom{k}{i}i! &= \sum_{i=0}^{k}\frac{L(L-1).....(L-i+1)}{i!}\times(i!)\times\binom{k}{i}\\ &\leq \sum_{i=0}^{k}L^i\binom{k}{i}\\ &= (L+1)^k\end{aligned}$$

Therefore, for a Q'_e satisfying (1), each edge h of $E_{mid(e)}$ is given a label $\lambda(h) \in (L+1)^k$. When $\lambda(h) = 0$, it indicates that no path exist on edge h, with no color scheme. For all the edges of $E_{mid(e)}$, each Q'_e satisfying (1) (and thus each candidate w.r.t. $E_{mid(e)}$) is identified by a unique label $\lambda \in (L+1)^{k|E_{mid(e)}|}$.

Algorithm ALG-PC first computes all candidates w.r.t. $E_{mid(e)}$ for every link e of T_C in a bottom-up way: For each leaf link $e = \{x, y\}$, the candidates are computed by enumeration: Since H_e is a single vertex, for any candidate Q_e w.r.t. $E_{mid(e)}$, Q''_e is empty. So we can find all candidates w.r.t. $E_{mid(e)}$ by enumerating all Q'_e satisfying (1). The labels and the associated candidates are kept in a table.

For an internal link e of T_C, let f and g be the children links of e. For each child link f and g, we have two dynamic programing tables. For each edge h of $E_{mid(f)}$ and $E_{mid(g)}$, every path on h is given a unique label. We denote by λ_e, λ_f, and λ_g the labels for the candidates w.r.t. $E_{mid(e)}$, $E_{mid(f)}$, and $E_{mid(g)}$, respectively. For every label λ_e, we compute the candidate associated to λ_e from the candidates associated to labels λ_f and λ_g. Let $X_1 = E_{mid(e)} \cap E_{mid(f)}$, $X_2 = E_{mid(e)} \cap E_{mid(g)}$, and $X_3 = E_{mid(f)} \cap E_{mid(g)}$.Then $X_i \cap X_j = \emptyset$ for $1 \leq i \neq j \leq 3$, $X_1 \cup X_2 = E_{mid(e)}$, $X_1 \cup X_3 = E_{mid(f)}$, and $X_2 \cup X_3 = E_{mid(g)}$. We say a label λ_e is formed from a label λ_f and a label λ_g if

- for $h \in X_1$, $\lambda_e(h) = \lambda_f(h)$,
- for $h \in X_2$, $\lambda_e(h) = \lambda_g(h)$, and
- for $h \in X_3$, $\lambda_f(h) = \lambda_g(h)$.

For a coloring λ_e formed from λ_f and λ_g, let $Q_e(f, g) = Q_f \cup Q_g$, where Q_f and Q_g are the candidates associated with λ_f and λ_g, respectively. The candidate associated to λ_e is the maximum $Q_e(f, g)$ for all pairs of λ_f and λ_g which form λ_e.

We use $\mathcal{Q}_e$ to denote the set of all subsets of feasible solution in P on $E_{mid(e)}$. For every $Q'_e \in \mathcal{Q}_e$, Q''_e is computed from Q''_f and Q''_g. Let $\mathcal{Q}'$ be a set of all possible subsets of feasible

solutions from the corresponding set of subsets Q'_f and set of subsets Q'_g and these paths are on the edges of $E_{mid(f)} \cup E_{mid(g)}$. For every $Q' \in \mathcal{Q}'$, let $Q'_e \subseteq Q'$, $Q'_f \subseteq Q'$ and $Q'_g \subseteq Q'$ are the set of feasible solution on some edges in $E_{mid(e)}$, $E_{mid(f)}$ and $E_{mid(g)}$, respectively. We initialize Q''_e to 0. If both f and g are the leaves of T_C then Q''_f and Q''_g are 0. For every Q', first we compute $|(Q'_f \cup Q'_g) \setminus Q'_e|$. Then the values $|(Q'_f \cup Q'_g) \setminus Q'_e|$,$Q''_f$ and Q''_g are added up. If this value is greater than the previous value of Q''_e, then Q''_e is updated to this value. Thus, Q'_e takes the maximum over all Q''_e.

Assume that Algorithm ALG-PC computes a carving-decomposition T_C of G with width γ in $f(n)$ time. In Step (II), Algorithm ALG-PC computes the candidates w.r.t. $E_{mid(e)}$ for every link e of T_C. For each internal link e of T_C, there are $(L+1)^{k|X_1|+|X_2|}$ or $(L+1)^{k|E_{mid(e)}|}$ possible subsets of partial solutions to store. Since $E_{mid(e)}$ is bounded by γ and each edge of $E_{mid(e)}$ has a load bounded by L and each subset of partial solution contains at most k paths on each edge of $E_{mid(e)}$. Therefore, there are at most $(L+1)^{k\gamma}$ possible subsets of partial solutions to store. While merging, we only need to consider the paths on X_1, X_2 and the paths on X_3. Since $|X_1 \cup X_2 \cup X_3| \leq 1.5\gamma$, there are at most $(L+1)^{k1.5\gamma}$ cases to consider. The time complexity to process one link is $O((L+1)^{k1.5\gamma}n)$ and the memory requirement is $O((L+1)^{k\gamma})$. The total time complexity and memory requirement are $O((L+1)^{k1.5\gamma}n^2 + f(n))$ and $O((L+1)^{k\gamma}n)$ respectively, where n represents number of vertices in the graph G. The running time and memory requirement are polynomial in the input parameters, when γ is bounded by a constant.

For a planar graph G, an optimal carving-decomposition T_C of G can be computed in $O(n^3)$ time [21, 39]. Algorithm ALG-PC solves the Max-PC problem in $O((L+1)^{1.5k\mathrm{cw}(G)}n^2 + n^3)$ time and $O(((L+1)^{k\mathrm{cw}(G)}n)$ memory space for G.

A ring is a well used topology in optical networks. The carving width of an undirected ring is 2 and a carving-decomposition with width 2 can be constructed in linear time in Step (I). Step (II) takes $O((L+1)^{3k}n)$ time and $O(((L+1)^{2k}n)$ memory space. Therefore for rings, Algorithm ALG-PC solves the Max-PC problem in $O((L+1)^{3k}n)$ time and $O((L+1)^{2k}n)$ memory space.

3.3 Exact algorithm for directed graphs

In many practical applications, communication links are directed and networks are modeled by directed graphs. Algorithm ALG-PC can be used to solve the Max-PC problem on

directed graphs as well. Let $\vec{G}$ be a directed graph and $H_{\vec{G}}$ be the underline graph of $\vec{G}$. In Step (I), we find a carving-decomposition T_C of small width for $H_{\vec{G}}$. In Step (II), we compute a solution of the problem using the dynamic programming approach based on T_C. For a link e of T_C, since each edge of the cut-set $E_{mid(e)}$ of $H_{\vec{G}}$ may correspond to two directed edges in $\vec{G}$, the cut-set of $\vec{G}$ corresponding to $E_{mid(e)}$ can have as many as $2|E_{mid(e)}|$ edges. From this, Algorithm ALG-PC solves the Max-PC problem in $O((L+1)^{3k\gamma}n^2+f(n))$ time, assume Step (I) finds a carving-decomposition of $H_{\vec{G}}$ with width γ in $f(n)$ time. For a simple directed planar $\vec{G}$, Algorithm ALG-PC solves the Max-PC problem in $O((L+1)^{3\mathrm{cw}(H_{\vec{G}})}n^2+n^3)$ time.

The directed ring consists of two directed cycles, one in the clockwise direction and the other in the counter clockwise direction. The Max-PC problem on directed rings can be solved on each of the cycles independently and the problem on each cycle can be viewed as the problem on the undirected ring. Therefore, our exact algorithm solves the Max-PC problem in $O((L+1)^{3k}n)$ time for the directed rings.

3.4 Approximation algorithm

When k is large, Algorithm ALG-PC may not be practical. One can observe that the Algorithm ALG-ED is a special case of ALG-PC, where $k = 1$. Using Algorithm ALG-ED as a subroutine, we can have an approximation algorithm for the Max-PC problem using the iterative greedy approach: Find a maximum edge-disjoint subset of P by Algorithm ALG-ED and assign the paths in the subset one color; remove the subset from P and the assigned color; repeat the above process until no color is available or all paths of P have been colored. It is shown in [40], the interactive greedy approach gives an approximation ratio of 1.58. Since the Algorithm ALG-ED solves the MEDPwPP problem, in $O((L+1)^{1.5\gamma}n^2+f(n))$ time for arbitrary graphs and in $O((L+1)^{1.5\mathrm{cw}(G)}n^2+n^3)$ time for planar graphs. Therefore, the approximation algorithm runs in $O((L+1)^{1.5\gamma}kn^2+f(n))$ time for arbitrary graphs, $O((L+1)^{1.5\mathrm{cw}(G)}kn^2+n^3)$ time for planar graphs and $O((L+1)^3kn)$ time for rings.

3.5 Approximation algorithm for directed graphs

Using Algorithm ALG-ED as a subroutine and using the iterative greedy approach, we can also have an approximation algorithm for the Max-PC problem for the directed graphs.

The algorithm runs in $O((L+1)^{3\gamma}kn^2 + f(n))$ time for arbitrary graph $\vec{G}$ and $O((L+1)^{3\mathrm{cw}(\vec{G})}kn^2 + n^3)$ time for simple planar graph $\vec{G}$. Since the directed ring consists of two directed cycles as described earlier, therefore, the approximation algorithm for directed rings runs in $O((L+1)^3kn)$ time.

3.6 Computational study

We generate the sets of paths as follows, given a positive integer α and an allowable maximum load $\mathcal{L}$ of the α paths. First generate α source-destination pairs in the given graph, randomly. Then find the shortest path between these pairs. If there are multiple shortest paths between source-destination nodes of a pair, then choose an arbitrary one. Consider only those shortest paths having path length greater than one. The load L of G is computed by using these set of paths. If this load L is more than $\mathcal{L}$, discard the generated paths and start over again. Note that: (i) The set of pairs may contain multiple source-destination pairs which have the same source and destination nodes or may have the same sets of paths, (ii) If $\mathcal{L}$ is small and α is large, it might not be possible to generate a set of α paths with maximum load $\mathcal{L}$.

We implemented the exact algorithm and the 1.58-approximation algorithm for the Max-PC problem and tested our implementations on the abstract models of a 16-node NSFNET (Figures 3.2), a 24-node ARPANET (Figure 3.3) and on the rings. We also implemented the 1.5-approximation algorithm (NPZ Algorithm) [33] for the Max-PC problem on rings. We also implemented the first-fit, random-fit, most-used, and least-used heuristics [2, 9, 32] for the Max-PC problem and tested these implementations on the NSFNET and the ring. For the NSFNET, we compared the results of our 1.58-approximation algorithm with the results of the first-fit, random-fit, most-used, and least-used heuristics. For the ring, we also implemented the 1.5-approximation algorithm (NPZ Algorithm) and compared the the results of our 1.58-approximation algorithm with the results of the NPZ Algorithm and with the results of the above mentioned heuristics. The computer used has an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ (2.4GHz) and 3GByte of internal memory. The operating system is SUSE Linux 10.2 and the programming language used is C++.

The computational results of the exact algorithm ALG-PC and 1.58-approximation algorithm for the NSFNET and ARPANET are reported in Tables 3.1 and 3.2, respectively. In the tables, k is the number of colors; $|P|$ is the number of paths; L is the maximum

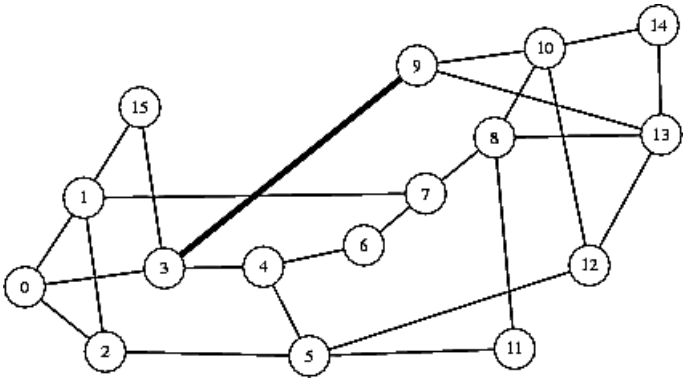

Figure 3.2: A 16-node NSFNET

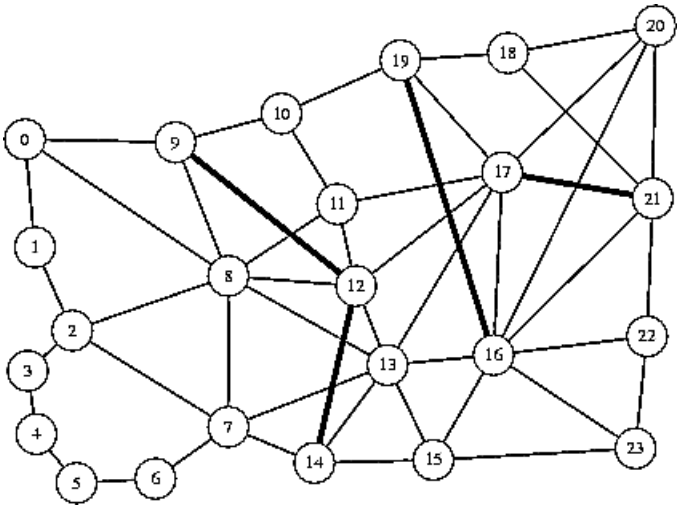

Figure 3.3: A 24-node ARPANET

number of paths on any edge; N_{opt}, t_{opt}, and M_{opt} are the number of paths found, the time used, and the memory space required by the exact algorithm ALG-PC, respectively; N_{app}, t_{app} and M_{app} are the number of paths found, the time used and the memory required by the approximation algorithm, respectively. The time is in seconds and the memory space is in MBytes, respectively. The time less than one second and memory less than 1 MBytes are denoted by < 1. We use symbol $*$ to denote the fact that no solution is obtained due to memory constraint. For each instance, we repeat the computation 7-10 times and report the average of these results. Notice that the NSFNET and ARPANET are not planar but very close to planar. In particular, removing the edges represented by bold segments in Figures 3.2 and 3.3 makes the graphs planar. For a non-planar graph G, we first compute a planar subgraph G' by removing the bold edges and then an optimal carving-decomposition T_C of G' is computed. We use T_C as the carving-decomposition of G. The width of T_C for G is at most $\mathrm{cw}(G') + 2$. More specifically, the width of T_C is 5 for NSFNET and 10 for ARPANET.

As shown in Table 3.1, the exact algorithm ALG-PC can solve the Max-PC problem on the NSFNET of $\gamma = 5$ for small k and L (e.g., $k = 3, L = 4$ and $k = 2, L = 8$) in a practical time and memory. When the load L is large, ALG-PC fails to solve the problem on a machine with 3GBytes memory. Algorithm ALG-ED can solve the MEDPwPP problem on the NSFNET for practical values of L. The ALG-ED based 1.58-approximation algorithm computes near optimal solutions for the Max-PC problem on the NSFNET for practical values of k and L.

The results in Table 3.2 show that ALG-PC can only solve the Max-PC problem on the ARPANET for $k = 2$ and $L = 4$. This is because the ARPANET has a larger carvingwidth $\gamma = 10$ than that of the NSFNET. Algorithm ALG-ED can solve the MEDPwPP problem for $L = 19$ in a practical time and memory space. The approximation algorithm gives near optimal solutions for the Max-PC problem with k and L as large as 16 and 15, respectively.

From Tables 3.1 and 3.2, we can see that the exact algorithm ALG-PC can solve the Max-PC problem for small k, L and γ in a practical time and memory space. If one of these parameters is large, ALG-PC may not be practical. In this case, the approximation algorithm is a good alternative. It gives solutions close to optimal, for practical values of k and L.

The computational results of 1.58-approximation algorithm and the first-fit, random-fit, most-used and least-used heuristics for the NSFNET are reported in Table 3.3. In this table,

Table 3.1: Results for the 16-node NSFNET backbone, where $\gamma = 5$

k	$\|P\|$	L	N_{opt}	t_{opt}	M_{opt}	N_{app}	t_{app}	M_{app}
2	20	4	12	< 1	< 1	11	< 1	< 1
3	20	4	15	3.915	261	15	< 1	< 1
4	20	4	*	*	*	18	< 1	< 1
2	25	6	11	1.756	55	11	< 1	< 1
4	25	6	*	*	*	19	< 1	< 1
2	30	8	15	8.067	907	12	< 1	< 1
4	30	8	*	*	*	23	< 1	< 1
2	90	18	*	*	*	17	< 1	< 1
4	90	18	*	*	*	33	< 1	< 1
8	90	18	*	*	*	55	< 1	< 1
16	90	18	*	*	*	73	< 1	< 1
1	200	49	8	4.88	210	8	2.893	171
2	200	49	*	*	*	19	4.77	748
4	200	49	*	*	*	35	7.793	1253
8	200	49	*	*	*	64	9.176	1523
16	200	49	*	*	*	100	10.495	1715
32	200	49	*	*	*	117	11.341	1753
1	350	64	10	7.45	1455	10	5.765	1010
2	350	64	*	*	*	19	10.908	2671
1	400	77	10	15.24	2999	10	9.996	2673

M_{ff}, M_{rf}, M_{mu} and M_{lu} are the number of paths colored by the first-fit, the random-fit, the most-used and the least-used heuristics, respectively. Other symbols are interpreted in a same way as that in Table 3.1, 3.2. Note that, for this table, we repeat the computation for each instance 5-7 times and report the average of the results. From Table 3.3, we can see that the approximation algorithm gives better solution than all heuristics.

Table 3.4 gives the results for the NSFNET when we view the NSFNET as the outline graph of a directed network with each edge of the graph corresponding to two directed links, one in each direction in the network. For this table, we repeat the computation for each instance 5-7 times and report the average of the results. Similar to the undirected graphs, Algorithm ALG-PC can solve the Max-PC problem when γ, k and L are small. The 1.58-approximation algorithm is an efficient alternative for the exact algorithm. The directed NSFNET network has carving width 10 and the undirected ARPANET network also has carving width 10. Since both directed NSFNET network and undirected ARPANET network have the same carvingwidth, therefore the results of directed NSFNET network can

Table 3.2: Results for the 24-node ARPANET backbone, where $\gamma = 10$

k	$\|P\|$	L	N_{opt}	t_{opt}	M_{opt}	N_{app}	t_{app}	M_{app}
2	20	4	15	9.875	1037	14	< 1	< 1
4	20	4	*	*	*	19	< 1	< 1
1	80	15	15	9.88	1010	15	7.36	783
2	80	15	*	*	*	28	12.245	1394
4	80	15	*	*	*	47	16.806	2142
8	80	15	*	*	*	63	20.004	2412
16	80	15	*	*	*	72	19.436	2410
1	100	19	18	38.11	3001	18	20.996	2859

be almost same as that of undirected ARPANET network.

The computational results of our exact algorithm ALG-PC, the 1.58-approximation algorithms for a ring of 30 nodes are reported in Table 3.5. This table also includes the results of the NPZ Algorithm, the first-fit and the random-fit heuristics. Notice that we only report the first-fit and the random-fit heuristics in the table because they color more paths than the least-used and most-used ones for the instances on the ring. In the table, N_{npz} is the number of paths colored by the NPZ Algorithm. Other symbols are interpreted in a same way as that in Table 3.1, 3.2 and 3.3. For these results, we repeat the computation for each instance 7-10 times and report the average of the results.

Because the ring has a small carvingwidth ($\gamma = 2$), the exact algorithm ALG-PC can solve the Max-PC problem on the ring for larger k and L (e.g., $k = 8, L = 6$; $k = 4, L = 10$ and $k = 2, L = 18$) in a practical time and memory space than those in NSFNET and ARPANET. Algorithm ALG-ED can solve the MEDPwPP problem on the ring for large L efficiently. The 1.58-approximation algorithm is also efficient for the Max-PC problem on the ring with practical values of k and L. The NPZ algorithm achieves the best known approximation ratio (1.5) for the Max-PC problem on the ring. The results show that the NPZ algorithm and the 1.58 approximation algorithm for the Max-PC problem on the ring has a similar performance in the number of paths colored. Both approximation algorithms color more paths than the heuristics. On the other hand, Our 1.58-approximation algorithm works for general graphs and has a comparable performance as that of the NPZ algorithm on the ring.

The run time and memory space of the algorithms increase very quickly when the path load L, the number of colors k or the carvingwidth increases, particularly for the exact

Table 3.3: Comparison of 1.58 approximation algorithm with the first-fit, random-fit, most-used and least-used heuristics for the 16-node NSFNET

k	$\|P\|$	L	N_{app}	N_{ff}	N_{rf}	N_{mu}	N_{lu}
2	100	17	19	19	18	19	17
4	100	17	33	33	33	33	32
8	100	17	55	54	52	54	52
16	100	17	77	69	68	69	68
17	100	17	78	69	69	69	69
2	150	30	20	20	20	19	20
4	150	30	35	34	35	33	35
8	150	30	64	59	58	56	55
16	150	30	89	81	79	79	78
32	150	30	101	84	84	84	84
40	150	30	104	84	84	84	84
44	150	30	104	84	84	84	84
1	250	45	12	11	11	11	11
2	250	45	23	22	22	22	22
4	250	45	40	38	38	38	38
8	250	45	64	59	61	59	61
16	250	45	104	90	90	90	91
32	250	45	142	112	112	112	112
40	250	45	146	112	112	112	112
44	250	45	160	112	113	112	112

algorithm ALG-PC. This coincides with the theoretical analysis: the exact algorithm is efficient for graphs with small carvingwidth when the load is not too large and number of colors used is small, but time and memory consuming if one of these parameters is large. The memory requirement seems a bottleneck for Algorithm ALG-PC to solve the Max-PC problems for large carvingwidth, large link load or large number of colors. The 1.58-approximate algorithm is more practical in time and memory requirement as compared to the exact algorithm and generate results close to the optimal ones.

Table 3.4: Results for a 16-node NSFNET backbone (directed case), where $2\gamma = 10$

k	$\|P\|$	L	N_{opt}	t_{opt}	M_{opt}	N_{app}	t_{app}	M_{app}
2	8	2	7	11.318	1310	7	< 1	< 1
2	10	3	10	21.415	2299	10	< 1	< 1
2	16	4	8	51.012	2881	8	< 1	< 1
1	60	9	15	33.34	2877	15	23.341	2223
2	60	9	*	*	*	33	30.764	2780
4	60	9	*	*	*	45	31.079	2891
8	60	9	*	*	*	50	32.214	2893
16	60	9	*	*	*	50	32.304	2890

Table 3.5: Comparison of the exact algorithm, 1.58 approximation algorithm, NPZ algorithm, first-fit and random-fit heuristics for a ring of 30 nodes

k	$\|P\|$	L	N_{opt}	N_{app}	N_{npz}	N_{ff}	N_{rf}
1	30	6	12	12	12	4	4
2	30	6	21	19	20	9	9
4	30	6	26	25	24	22	22
6	30	6	28	28	27	27	27
8	30	6	30	29	29	27	27
1	60	10	15	12	12	4	4
2	60	10	25	23	24	9	9
4	60	10	40	39	40	22	22
8	60	10	*	53	53	27	27
16	60	10	*	59	60	27	27
1	120	18	15	13	12	10	10
2	120	18	25	23	24	15	15
4	120	18	*	47	48	24	23
8	120	18	*	80	81	27	27
16	120	18	*	103	105	27	27
32	120	18	*	118	120	27	27
1	240	47	22	19	19	9	9
2	240	47	*	26	25	18	18
4	240	47	*	36	37	30	31
8	240	47	*	70	72	45	47
16	240	47	*	135	134	72	70
32	240	47	*	192	194	79	79
40	240	47	*	206	206	79	79
44	240	47	*	206	210	80	80

Chapter 4

Algorithm for the MEDBCS Problem

In this chapter, we describe an implementation of a bidimensionality theory based algorithm for the maximum edge degree-bounded connected subgraph (MEDBCS) problem, which is described in [38]. The MEDBCS problem is a generalization of the longest path or longest cycle problem where $d \leq 2$ and $d = 2$, respectively. In [38], the bidimensionality theory based algorithm for the MEDBCS problem is discussed, however the dynamic programming step is not discussed in detail. We give a detailed description of the dynamic programming step of the bidimensionality theory based algorithm. We also perform a computational study of the algorithm for planar graphs.

4.1 Non-crossing property

A *non-crossing partition* is a partition $\tau(h) = \{\tau_1, ..., \tau_m\}$ of a cyclically ordered set $S = \{1, ..., h\}$ such that there are no numbers $a < b < c < d$ where $a, c \in \tau_i$, and $b, d \in \tau_j$ with $i \neq j$. A partition can be visualized by a circle with n equidistant vertices on its border, where every set of the partition is represented by the convex polygon with its elements as endpoints. A partition is non-crossing if these polygons do not overlap. Non-crossing partitions were introduced by Kreweras [27], who showed that the number of non-crossing partitions over n vertices is equal to the n-th Catalan number:

$$CN(n) = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}} \approx 4^n \tag{4.1}$$

In [12], it is proved that, dynamic programing algorithms for a planar graph can be speed up by using a non-crossing property.

4.2 Algorithm for the MEDBCS problem

A *parameter* P is a function mapping graphs to positive integers. A problem is *bidimensional* if the value of the parameter P depends on the size of the grid and $P(H) \leq P(G)$, where H is minor of G. The aim is to decide if $P(G) \leq k$ for given G and k. Demaine *et al.* in [11] defined bidimensionality as: A parameter P is *minor bidimensional* with density δ if P is closed under taking minors, for $(r \times r)$-grid R, $P(R) = (\delta r)^2 + o((\delta r)^2)$. A parameter P is *contraction bidimensional* with density δ if P is closed under contractions, for any partially triangulated $(r \times r)$-grid R, $P(R) = (\delta_R r)^2 + o((\delta_R r)^2)$. The parameter P is called *bidimensional* either it is minor bidimensional or contraction bidimensional. Notice that a minor bidimensional problem is also a contraction bidimensional but the inverse may not be true. The density is usually $0 < \delta \leq 1$. For example a vertex cover problem. A parameter vertex cover is minor-bidimensional and for a $(r \times r)$-grid, the size of vertex cover is at least $r^2/2$. Therefore, a parameter vertex cover has density $1/\sqrt{2}$.

The *maximum edge degree-bounded connected subgraph* (MEDBCS) problem is an example of bidimensional problems and one of the classical NP-hard problems [18]. The MEDBCS problem is a generalization of the longest path problem, where $d \leq 2$ and the Hamiltonian cycle problem, where $d = 2$. The bidimensionality theory based subexponential parametrized algorithm for the MEDBCS problem is given by Sau and Thilikos in [38]. It is proved in [38] that the parameter for the MEDBCS problem is bidimensional by showing that the parameter the MEDBCS is minor closed and from the following lemma.

Lemma 4.2.1 *[38] For any $d \geq 2$ and for any planar graph G it holds that $bw(G) \leq 3/\delta.\sqrt{MEDBCS(G)} + O(1)$, with $\delta = 1$, if $d = 2$, $\delta = \sqrt{3/2}$, if $d = 3$ and $\delta = \sqrt{2}$, if $d \geq 4$.*

Once it is the proved that the problem is bidimensional, next step is to solve the problem by using a bidimensionality theory based algorithm. A bidimensionality theory based algorithm has two major steps. Step (I) is to find the branch-decomposition and Step (II) is

to check either bw(G) is larger or smaller than some threshold value. If bw(G) is larger than some threshold value then $P(G)$ is computed from $gm(G)$. Otherwise, $P(G)$ is computed optimally by the dynamic programing approach. For the MEDBCS problem, in Step (II), we have to check either bw(G) $> 3\sqrt{k}/\delta$, where δ is a constant depending on the bounded degree d: for $d = 2$ $\delta = 1$, for $d = 3$ $\delta = \sqrt{3/2}$ and for $d \geq 4$ $\delta = \sqrt{2}$. If bw(G) $\leq 3\sqrt{k}/\delta$, then the value of the MEDBCS is computed, by using a dynamic programing algorithm, in $O(2^{log(5(d+1))6\sqrt{k}/\delta}\sqrt{k}n)$ time [38]. Otherwise, the value of the MEDBCS is computed from $gm(G)$ or $cm(G)$.

For the MEDBCS problem, an input instance to the algorithm consists of a graph G, a degree d. Let the subset $\mathcal{E} \subseteq E(G)$ is a *feasible solution* such that the subgraph having the edge set $\mathcal{E}$ is connected and degree of the vertices in the subgraph is bounded by d. The algorithm outputs a maximum feasible solution. For Step (I) we find the branch decomposition. In [39], it is proved that for a planar graph G of branchwidth h, there exists an sc-decomposition of G with width h.

Theorem 4.2.1 *[39] Let G be a planar graph of branchwidth at most h without vertices of degree one embedded on a sphere. Then there exists an sc-decomposition of G of width at most h.*

It is further shown that an optimal sc-decomposition can be computed in $O(n)^3$ time [21, 39]. For computing bw(G) and an optimal branch-decomposition of G, we use the tools reported in [5, 6]. After finding bw(G), we check that either bw(G) $< 3\sqrt{k}/\delta$, if it is not, then the answer to the MEDBCS problem is "NO" and we find the value of the MEDBCS parameter from $cm(G)$. Since a $(g \times h)$-cylinder contains a $(r \times r)$-grid as a minor, therefore, the value of $cm(G)$ is computed from large $(g \times h)$-cylinder minors. For computing large cylinder minors, we use the tool reported in [41]. This tool finds the $(g \times h)$-cylinder with $g \geq$ bw(G)/2. Otherwise, the MEDBCS problem is solved by using the dynamic programing approach.

Branch-decomposition based algorithms are proposed for special cases of the MEDBCS problem, the longest path and Hamiltonian cycle problems, in planar graphs [12]. A bidimensional algorithm for the general MEDBCS problem is discussed in [38] without much detail of the dynamic programming step. Next, we give a detailed description of the dynamic programming step for the algorithm in [38] for the MEDBCS problem. Much of the description is based on the algorithm given in [12].

In Step (II), for the dynamic programing approach, the branch decomposition T_B is first converted to a rooted binary tree by replacing an internal link $e = \{x, y\}$ with three $\{x, z\}$, $\{z, y\}$, $\{r, z\}$ links. In these links z and r are new nodes added in T_B, where r is the root and $\{z, r\}$ is an internal link. For every internal link e of T_B, e has two child links f and g incident to e. Let T_e be the subtree of T_B consisting of all the descendant links of e and T'_e be the subtree of T_B consisting of all the rest of the links which are not descendant of e. Let G_e be the subgraph of G induced by the edges at the leaf nodes of T_e and G'_e be the subgraph of G induced by the edges at the leaf nodes of T'_e. The dynamic programming step finds the partial solutions of G_e for every subtree T_e of T_B from leaves to the root in a bottom-up way.

For an internal link e of T_B and G_e, that is the subgraph of G induced by the edges associated with the leaves of the T_e, let O_e be the corresponding noose in Σ. The noose O_e partitions Σ into two discs and separate G_e from G'_e. Let $V(O_e)$ denote the set of vertices on O_e, which is $= V_{mid(e)}$. Each child link f and g has a corresponding noose. Let O_f and O_g be the noose corresponding to the separation induced by the links f and g, respectively. Let $V(O_f)$ and $V(O_g)$ denote the set of vertices on O_f and O_g, respectively, which are $= V_{mid(f)}$ and $= V_{mid(g)}$, respectively. The set $V_{mid(f)} \cup V_{mid(g)} \cup V_{mid(e)}$ is partitioned into four subsets as illustrated in Figure 4.1:

$$X_1 = V_{mid(e)} \backslash V_{mid(g)}$$

$$X_2 = V_{mid(e)} \backslash V_{mid(f)}$$

$$X_3 = V_{mid(e)} \cap V_{mid(g)} \cap V_{mid(g)}$$

$$X_4 = (V_{mid(f)} \cap V_{mid(g)}) \backslash V_{mid(e)}$$

Given a labeling $\mathcal{L}_e : E(G_e) \rightarrow \{0, 1\}$, on each edge of G_e. For every vertex v of G_e, we define the $deg_{\mathcal{L}e}(v)$ to be the sum of the labels on the edges incident to v. Let $G_{\mathcal{L}e}$ be the subgraph induced by the edges with label '1'. Let $CG_{\mathcal{L}e} = \{CG_{\mathcal{L}e1}.....CG_{\mathcal{L}ek}\}$ be the set of connected components in $G_{\mathcal{L}e}$. $\mathcal{L}_e$ is called a valid partial solution of the MEDBCS problem if all the edges of $CG_{\mathcal{L}ei} \in CG_{\mathcal{L}e}$with label '1' are connected and $deg_{\mathcal{L}e}(v)$ for each vertex of the $CG_{\mathcal{L}ei}$ is $\leq d$. For instance, for a partial solution of the MEDBCS problem with $d \leq 2$, each connected component of $G_{\mathcal{L}e}$ is a cycle if $deg_{\mathcal{L}e}(v)$ is exactly 2 for all the vertices, or a path if $deg_{\mathcal{L}e}(v)$ is ≤ 2 . The *size* of the partial solution is defined as the number of edges with label '1' in $CG_{\mathcal{L}e}$ such that $deg_{\mathcal{L}e}(v)$ for each vertex of $CG_{\mathcal{L}e}$ is bounded by d.

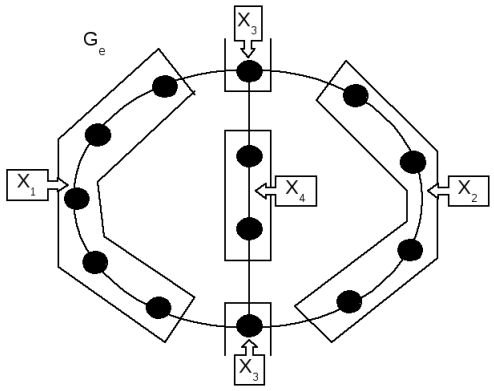

Figure 4.1: Subsets of cut sets $V_{mid(f)}$, $V_{mid(g)}$ and $V_{mid(e)}$

Let π be the clockwise or counter clockwise order of vertices of $V_{mid(e)}$, starting with an arbitrary vertex. We color each vertex $v \in V_{mid(e)}$, in the order π. The color assignment is done by using following five basic colors.

- 0: v does not exist in any connected component.
- $1_{[}$: v is the starting vertex of the connected component as computed in the order π.
- $1_{]}$: v is the ending vertex of the same connected component in which $1_{[}$ is found, and it is also computed according to the order π.
- $1^{\star}$: v is the somewhere in the middle of the connected component which has already $1_{[}$ and $1_{]}$ vertics.
- 1: v is the only one vertex which is found in the order π for the connected component.

Since $deg_{\mathcal{L}e}$ for each vertex of $V_{mid(e)}$ is bounded by degree d, therefore each of the basic colors except 0, is further categorized by degree d. Therefore, each of the basic color is further assigned colors from $1, 2.....d$. For example, for $1_{[}$, there would be set of d colors $\{1_{[1}, ...1_{[d}\}$. For color 1, there could be the possibility that only one vertex exist with $deg_{\mathcal{L}e}(v)$ zero. For four basic colors, where each color is bounded by degree d, there are $\{1_{[1}, ...1_{[d}\}$ set of d colors for each basic color, therefore, total number of colors are $(4 \times d) + 2$.

During dynamic programing, for every internal link e, all possible partial solutions are computed based on $E_{mid(f)}$ and $E_{mid(g)}$. For an internal link e, while merging the solutions, let f and g be the child links, which are already processed. For a link e and the corresponding noose O_e, the state of the dynamic programing is specified by an ordered l-tuple $t_e := (v_1, ...v_l)$. The variables $v_1, ..., v_l$ correspond to the vertices of $O_e \cap V(G)$ taken according to the cyclic order π. Each of v_i takes one of the $(4 \times d) + 2$ colors. Hence, there are at most $O(((4 \times d) + 2)|V(G)|)$ states. For every state, a value $\mathcal{E}_e$ is computed which keeps the maximum size of the valid partial solutions that can be represented by this state.

We perform the dynamic programming over the middle set $V_{mid(e)}$, starting at the leaves of T_B and working bottom-up towards the root edge. For each leaf link of T_B, the corresponding edge of G has two adjacent vertices of G. We set the value of $\mathcal{E}_e = 1$, if both vertices exist with $deg_{\mathcal{L}e}(v) = 1$, otherwise the value of $\mathcal{E}_e = 0$. For each internal link, we merge the partial solutions. The connected components which intersect are fused and the degrees of the vertices are updated. For each edge of the branch decomposition, the tables of our dynamic programing algorithm store all the partial solutions to the problem in the graph processed so far. Partial solutions may have several connected components, therefore, we need to keep track of them too. Along with, we also need to control the degrees of the vertices in the partial solutions, in order to assure that maximum degree of the output subgraph is bounded by d. At the root node, we get the output subgraph that should be connected and bounded by d. Let $CG_{\mathcal{L}f}$ and $CG_{\mathcal{L}g}$ be the set of components from child links f and g respectively. For merging the components, there are two steps. First, we have to check either the specific component $CG_{\mathcal{L}fi} \in CG_{\mathcal{L}f}$ can be merged with $CG_{\mathcal{L}gj} \in CG_{\mathcal{L}g}$ or not. If it can be merged then in the second step, we have to check, either the $deg_{\mathcal{L}e}(v)$ for all the vertices of $CG_{\mathcal{L}em} \in CG_{\mathcal{L}e}$ is bounded by d or not. If both the conditions are true then merge the specific components. Suppose $CG_{\mathcal{L}fi} \in CG_{\mathcal{L}f}$ can be merged with $CG_{\mathcal{L}gj} \in CG_{\mathcal{L}g}$. While merging these components, follow the following merging rules.

$\deg_{\mathcal{L}e}(v) = deg_{\mathcal{L}f}(v)$, if $v \in X_1$

$deg_{\mathcal{L}e}(v) = deg_{\mathcal{L}g}(v)$, if $v \in X_2$

$deg_{\mathcal{L}e}(v) = deg_{\mathcal{L}f}(v) + deg_{\mathcal{L}f}(v)$, if $v \in X_3, X_4$

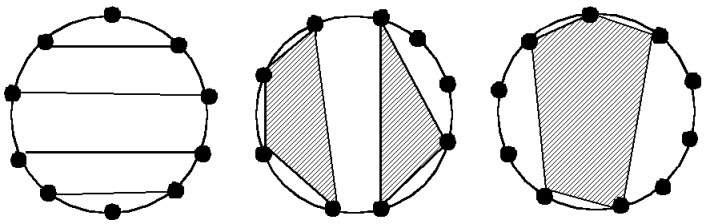

Figure 4.2: Catalan structures in the $V_{mid(e)}$ of a sc-decomposition

After merging the components, find a state by assigning colors to the vertices of $V_{mid(e)}$ according to the cyclic order π and degrees of vertices. If the corresponding state has already some solution set, then compare the recent results with the previous one. It is important to mention here, that while comparing the results, we have to compare all the components. After comparing, for the specific state, keep the solution having maximum number of edges in all the components.

Sau and Thilikos in [38] proved that the MEDBCS problem can be solved in $(d+1)^{2h}.2^{4h.logh}.h.n$ steps for an arbitrary graph G of branchwidth at most h. For detailed analysis of the algorithm readers are referred to [38]. When the input graph G is restricted to a planar graph, the running time can be reduced by using the sc-decomposition and non-crossing property [12]. Non-crossing partitions speed up the dynamic programming approach when it is applied to planar graphs. For the MEDBCS problem, when the input graph G is restricted to be planar, then the subgraph (connected components) obtained by the intersection of a partial solution of the child links is also planar. Each connected component can be drawn inside a cycle such that they touch the cycle on its vertics and they do not intersect, as illustrated in Figure 4.2. Since the number of non-crossing partitions over $V_{mid(e)}$ vertices, is equal to the Catalan number equation 4.1. Therefore, for every planar graph G, the MEDBCS problem is computed in $O((d+1)^{2h}.5^{2h}.h.n)$ steps and for any $d \geq 2$, the MEDBCS problem is solvable is $O(2^{log(5(d+1))6\sqrt{k}/\delta}\sqrt{k}n + n^3)$ time [38] time.

Table 4.1: Computational results of dynamic programming algorithm for the MEDBCS problem

$CLASS$	$\|V(G)\|$	$\|E(G)\|$	bw(G)	d	$\|Max-E\|$	t_{Max-E}	M_{Max-E}
I	350	1044	4	2	287	2.215	512
				3	435	18.16	1262
	500	1494	4	2	363	3.613	1050
				3	623	60.819	2640
	800	2394	4	2	578	8.403	2644
				3	*	*	*
	900	2694	4	2	*	*	*
II	50	139	5	2	50	0.955	28
				3	74	2.98	122
	100	288	6	2	100	5.16	238
				3	149	11.53	1414
III	153	248	4	2	131	0.249	56
				3	194	4.948	87
	257	433	5	2	225	1.67	727
				3	323	11.10	1325
	495	852	5	2	426	9.751	295
				3	698	15.98	1255

4.3 Computational study

We test the bidimensionality theory based algorithm for the MEDBCS problem on three classes of graphs. Class I is a collection of random maximal planar graphs which are generated by LEDA [30]. Class II is a collection of triangulation graphs which are also generated by LEDA. Class III is a collection of random planar graphs which are generated by PIGALE library [10].

The computer we used has an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ (2.4GHz) and 3GByte of internal memory. The operating system is SUSE Linux 10.2 and the programming language used is C++.

The computational results of the dynamic programing algorithm for the above mentioned three classes of graphs are reported in Table 4.1. In the table, for the given instance, $|V(G)|$ is the number of vertices in the given graph, $|E(G)|$ is the number of edges in the given graph G, bw(G) is the branchwidth of the given graph, d is the input degree, $|Max-E|$ is the maximum number of edges obtained for the MEDBCS problem, t_{Max-E} is the time

to solve the MEDBCS problem for the given graph and M_{Max-E} is the memory required by the algorithm. It is necessary to mention here that time is in seconds and memory is in MBytes. We use symbol $*$ to denote the fact that no solution is obtained due to memory constraint. Note that, we repeat the computation five times for every particular instance and only report the best possible result.

As shown in Table 4.1, the algorithm can solve the MEDBCS problem on graphs with small bw(G), as in class I, in a practical time and memory. When d is 2, the algorithm can solve the problem for large graphs efficiently. When d is increased to 3, the running time increases and more memory is used as compared to $d = 2$ and the algorithm cannot solve the problem for larger graphs even when the branchwidth is 4. For class II, the graphs have branchwidth larger than other classes I and III, the algorithm can solve the problem within practical time and memory for smaller graphs. For larger graphs, the algorithm can solve the problem for $d = 2$, but it quickly goes out of memory, when $d > 2$. Therefore, the running time of the algorithm depends on the branchwidth of the graph, the size of the graph and on d.

It is necessary to mention here that, we get the results for $d = 2$ and $d = 3$, because when $d > 3$, the running time increases very quickly and also due to limited available memory, we cannot get the results even for graphs of smaller size. However, if the memory is not the issue, our algorithm can solve the MEDBCS problem even for large d. When either the bw(G) or d is large, the algorithm fails to solve the problem on a machine with 3GBytes memory. In conclusion, the bidimensionality theory based algorithm for the MEDBCS problem can solve the problem, for graphs with small bw(G) and d, in practical time and memory space.

When the branchwidth, bw(G) of the given planar graph G is large, the dynamic programing algorithm is not practical for solving the MEDBCS problem. According to the bidimensionality theory, we can find an approximate solution of the MEDBCS problem from the largest grid minor. For computing the largest grid minors, we use the tool reported in [41], named GT tool. The GT tool is an implementation of the algorithm designed by Gu and Tamaki in [22], to compute an approximation of the largest cylinder minor. The GT tool finds the $(g \times h)$-cylinder minor, where $g \geq 3$ and $h \geq 1$. $cm(G)$ denote the largest integer g such that G contains a $(g \times g/2)$-cylinder as a minor. It is shown in [22] that for a planar graph G, a $(g \times g/2)$-cylinder minor with $g \geq bw(G)/2$, can be computed efficiently. Therefore, for a given planar graph G with large bw(G), we can compute a large

cylinder minor of G. From the large cylinder minor of G, we can find a solution of the MEDBCS problem with the number of edges close to the number of edges in the cylinder minor. Using cylinder minors, we can find a lower bound on the size of solutions for the MEDBCS problem. For the MEDBCS problem with $d \leq 2$, the problem is a generalization of the longest path problem, a $(g \times h)$ cylinder minor of G implies a path of length $\geq (g \times h) - 1$. The MEDBCS problem with d exactly 2, the problem is a generalization of the Hamiltonian cycle problem, a $(g \times h)$ cylinder minor of G implies a cycle of length $\geq (g \times h)$. For the MEDBCS problem with d at most 3, and $d \geq 4$, a $(g \times h)$-cylinder minor of G implies the solution of size $\geq [\{g(h-1)+2\} + \{h(g-1) - \lfloor \frac{g-1}{2} \rfloor (h-2)\}]$ and $\geq [\{g(h-1))+(h(g-1)\}]$, respectively. Therefore, for the MEDBCS problem with d exactly 2, the lower bound is $(g \times h)$, for $d \leq 2$, the lower bound is $(g \times h) - 1$ and for d at most 3, the lower bound is $[\{g(h-1)+2\} + \{h(g-1) - \lfloor \frac{g-1}{2} \rfloor (h-2)\}]$. For $d \geq 4$, the lower bound is $[\{g(h-1)) + (h(g-1)\}]$.

It is necessary to mention here that GT tool generates different results at every execution, because it starts from an arbitrary edge. For branchwidth at most 6, we can find the exact solution of the MEDBCS problem by using the dynamic programing approach, in practical time and memory, as shown in Table 4.1. In the case of branchwidth > 6, we use GT tool to find an approximate solution for the MEDBCS problem. We use the GT tool on classes II and III, mentioned above, because it contain graphs of large branchwidth 7 and 8. The results are reported in Table 4.2, where $cm(G)$ is the size of the cylinder minor found by GT tool. LB is the lower bound for the solution of the MEDBCS problem implied by the cylinder minor. Other symbols are interpreted in the same way as in Table 4.1. In order to get an approximation of the largest grid minor, we run the GT tool for each graph instance many times.

Table 4.2: Computational results of GT tool for the MEDBCS problem

$CLASS$	$\|V(G)\|$	$\|E(G)\|$	bw(G)	$cm(G)$	d	LB
II	1000	1491	8	(6×5)	≤ 2	29
					$=2$	30
					≤ 3	45
	2000	5977	8	(6×4)	≤ 2	23
					$=2$	24
					≤ 3	36
III	2009	3369	7	(7×4)	≤ 2	27
					$=2$	28
					≤ 3	41
	3586	6080	8	(6×5)	≤ 2	29
					$=2$	30
					≤ 3	45

Chapter 5

Conclusion and Future Work

In this thesis, we focus on two graph optimization problems, maximum path coloring (Max-PC) problem and maximum edge degree-bounded connected subgraph (MEDBCS) problem. We use two graph decomposition methods, branch-decomposition and carving-decomposition, to design and describe the algorithms for the problems. For the Max-PC problem, we proposed a carving-decomposition based exact algorithm ALG-PC. Based on this algorithm, we also developed 1.58-approximation algorithms for the problem. We also performed a computational study of both of the algorithms. The computational results show that the exact algorithm is practical when the given carving-decomposition has small width γ, the number of colors k is small and the link load L is not large. The approximation algorithm give solutions close to optimal ones and is an efficient alternative for the exact algorithm when one of γ, k and L is large. Both of the algorithms not only work for undirected graphs but also for directed graphs. The computational study shows that the memory requirement is a major bottleneck for our algorithms to solve the problem instances with large γ, k or L. We also implemented the well used first-fit, random-fit, most-used and least used heuristics for the Max-PC problem and compared the results with our approximation algorithm. We found that our approximation algorithm always gives better solution than the heuristics. We also implemented the 1.5-approximation algorithm (NPZ Algorithm) for the ring network and compared the results of our 1.58-approximation algorithm with those of NPZ Algorithm. It is shown that the 1.58-algorithm has a similar performance as that of NPZ Algorithm. Both algorithms find solutions close to optimal ones and are efficient for the Max-PC problem with practical values of k and L on the ring network. Both NPZ Algorithm and the 1.58-approximation algorithm color more paths than the heuristics studied.

It is worth to consider how to reduce the memory requirement by the algorithms.

For MEDBCS problem, we implemented a bidimensionality theory based algorithm described in [38]. The bidimensionality theory based algorithm has two major ingredients branch-decomposition and the largest grid minor. We implemented a branch-decomposition based algorithm for MEDBCS problem and give a detailed description of the dynamic programing step. We also perform a computational study of the algorithm on planar graphs and found that practical results coincide with the theoretical results. The computational results show that the algorithm can solve the problem for graphs with small branchwidth and with small d in practical time and memory. Our algorithm can also solves the problem for larger instances if memory is not a hurdle and the running time might be in hours. One may find new techniques to reduce the memory usage of the algorithm. We performed a computational study in planar graphs, one can extend the implementation for arbitrary graphs.

Bibliography

[1] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh. Degree-constrained subgraph problems: Hardness and approximation results. In *Proceedings of Approximation and Online Algorithms, 6th International Workshop, WAOA, Karlsruhe, Germany*, volume 5426 of *Lecture Notes in Computer Science*, pages 29–42, 2008.

[2] R. A. Barry and P. A. Humblet. Models of blocking probability in all-optical networks with and without wavelength changers. *IEEE Journal on Selected Areas in Communications*, 14(5):858–867, 1996.

[3] M. Bashir and Q.P. Gu. Carving-decomposition based algorithms for the maximum path coloring problem. In *To be appeared in IEEE international conference on communications*, ICC'12, 2012.

[4] Z. Bian. *Algorithms for wavelength assignment and call control in optical networks.* PhD thesis, Simon Fraser University, 2008.

[5] Z. Bian and Q.P. Gu. Computing branch decomposition of large planar graphs. In *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, pages 87–100, 2008.

[6] Z. Bian, Q.P. Gu, M. Marzban, H. Tamaki, and Y. Yoshitake. Empirical study on branchwidth and branch decomposition of planar graphs. In *ALENEX*, pages 152–165, 2008.

[7] M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3):225–235, 1995.

[8] L. S. Chandran and F. Grandoni. Refined memorization for vertex cover. *Information Processing Letters*, 93:125–131, 2005.

[9] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: an approach to high bandwidth optical WAN's. *IEEE Transactions on Communications*, 40(7):1171–1182, 1992.

[10] H. de Fraysseix and P. O. de Mendez. PIGALE-public implementation of a graph algorithm library and editor,. *SourceForge Project page http://pigale.sourceforge.net/*, 2008.

[11] 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*, 52:866–893, 2005.

[12] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

[13] T. Erlebach. *Scheduling Connections in Fast Networks.* PhD thesis, Technische Universitat Munchen, 1999.

[14] T. Erlebach and K. Jansen. Maximizing the number of connections in optical tree networks. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, ISAAC '98, pages 179–188, 1998.

[15] T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1-2):33–50, 2001.

[16] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001.

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[18] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[19] M. Garey, D. Johnson, G. Miller, and C. Papadimitiou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1(2):216–227, 1980.

[20] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

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

[22] Q. P. Gu and H. Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. In *ISAAC (2)*, pages 85–96, 2010.

[23] U. I. Gupta, D. T. Lee, and J. Y-T Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982.

[24] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[25] C. Kaklamanis and G. Persiano. Efficient wavelength routing on directed fiber trees. In *Proceedings of the Fourth Annual European Symposium on Algorithms*, ESA '96, pages 460–470, 1996.

[26] C. Kaklamanis, P. Persiano, T. Erlebach, and K. Jansen. Constrained bipartite edge coloring with applications to wavelength routing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, ICALP '97, pages 493–504, 1997.

[27] G. Kreweras. Sur les partitions non croisees d'un cycle. *Discrete Mathematics*, 1(4):333–350, 1972.

[28] V. Kumar and E. J. Schwabe. Improved access to optical bandwidth in trees. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 437–444, 1997.

[29] L. Lovász and M. Plummer. *Matching theory*. Annals of discrete mathematics. North-Holland, 1986.

[30] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[31] M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 548–557, 1995.

[32] A. Mokhtar and M. Azizoglu. Adaptive wavelength routing in all-optical networks. *IEEE/ACM Transactions on Networking*, 6:197–206, 1998.

[33] C. Nomikos, A. Pagourtzis, and S. Zachos. Satisfying a maximum number of pre-routed requests in all-optical rings. *Computer Networks*, 42:55–63, 2003.

[34] P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 134–143, 1994.

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

[36] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004.

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

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

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

[40] P.J. Wan and L. Liu. Maximal throughput in wavelength-routed optical networks. *Discrete Mathematics and Theoretical Computer Science*, 46:15–26, 1998.

[41] C. Wang and Q. P. Gu. Computational study on bidimensionality theory based algorithm for longest path problem. In *Proceedings of the 22nd International Symposium on Algorithms and Computation*, pages 364–373, 2011.