COMPUTATIONAL STUDY FOR DOMINATION PROBLEMS IN PLANAR GRAPHS

by

Marjan Marzban

B.Sc., Teacher Training University, 1999

M.Sc., Tarbiat Moddaress University, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2012

APPROVAL

Name:	Marjan Marzban
Degree:	Doctor of Philosophy
Title of thesis:	Computational Study for Domination Problems in Planar Graphs
Examining Committee:	Dr. Wo-Shun Luk Chair
	Dr. Qianping Gu, Senior Supervisor Professor
	Dr. Mohammed Hefeeda, Supervisor Associate Professor
	Dr. Jiangchuan Liu, SFU Examiner Associate Professor
	Dr. Caoan Wang, External Examiner Professor, Computer Science, Memorial University of Newfoundland
Date Approved:	March 5th, 2012

Declaration of
Partial Copyright Licence

Abstract

The DOMINATING SET problem is one of the most widely studied problems in graph theory and networking. For a graph $G(V, E)$, $D \subseteq V(G)$ is a dominating set of G if each vertex v of G is either in D or has a neighbour in D. Finding a minimum dominating set for arbitrary graphs is NP-hard and remains NP-hard for planar graphs. Recently, based on the notion of branch-decompositions, there has been significant theoretical progress towards fixed-parameter algorithms and polynomial time approximation schemes (PTAS) for the problem in planar graphs. However, little is known on the practical performances of those algorithms and a major hurdle for such evaluations is lack of efficient tools for computing branch-decompositions of input graphs. We develop efficient implementations of algorithms for computing optimal branch-decompositions of planar graphs. Based on these tools, we perform computational studies on a fixed-parameter exact algorithm and a PTAS for the DOMINATING SET problem in planar graphs. Our studies show that the fixed parameter exact algorithm is practical for graphs with small branchwidth and the PTAS is an efficient alternative for graphs with large branchwidth. We also perform analytical and computational studies for a branch-decomposition based fixed parameter exact algorithm for the CONNECTED DOMINATING SET (CDS) problem in planar graphs. We prove a better upper bound for the branchwidth in terms of the minimum size of CDS. Using this improved upper bound, we achieve an improved time complexity for the exact algorithm for the CDS problem. Finally, we show that the density of the CDS problem in planar graphs is $\frac{1}{\sqrt{5}}$ in bidimensionality theorem.

To Alireza

To my parents.

"When I want to understand what is happening today or try to decide what will happen tomorrow, I look back."

— Omar Khayyam, 10481131.

Acknowledgments

I am greatly thankful to my senior supervisor Dr. Qianping Gu. I appreciate all his contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating. The joy and enthusiasm he has for his research was motivational for me, even during tough times in the Ph.D. pursuit. I would like to thank Dr. Mohamed Hefeeda for being my supervisor. I am thankful to Dr. Jiangchuan (JC) Liu and Dr. Caoan Wang for being my examiners. I am also grateful to my colleague Dr. Zhengbing Bian for his collaborations and helps.

I am deeply thankful to my parents for their unconditional love and support in every moment of my life.

Last but not least, a big thank you to my love Alireza. It has always been wonderful and comforting to have him beside me. Without his love, understanding and endless supports, it would have been impossible for me to finish this thesis.

Contents

Approval **ii**

Abstract **iii**

Dedication **iv**

Quotation **v**

Acknowledgments **vi**

Contents **vii**

List of Tables **xi**

List of Figures **xiii**

1 Introduction **1**

1.1 DOMINATING SET problem 3

1.2 CONNECTED DOMINATING SET problem 5

1.3 Basic techniques 6

1.3.1 Branch-decomposition of planar graphs 7

1.3.2 Outer planar-decomposition 7

1.3.3 Kernelization 8

1.4 Motivation and contribution 8

1.5 Organization 11

2 Preliminaries and related works **12**
2.1 Preliminaries . . . 12
2.1.1 Basic definition . . . 12
2.1.2 Branch-decompositions and grid-minors . . . 14
2.1.3 Domination problems . . . 17
2.2 Previous works . . . 17
2.2.1 Branch-decomposition . . . 18
2.2.2 DOMINATING SET problem . . . 20
2.2.3 CONNECTED DOMINATING SET (CDS) problem . . . 22
2.2.4 PTAS and heuristics for DOMINATING SET problem . . . 23

3 Branch-decomposition of planar graphs **26**
3.1 Optimal branchwidth of planar graphs . . . 26
3.1.1 Rat-catching game . . . 28
3.1.2 Seymour and Thomas algorithm . . . 29
3.2 Observations for improvements . . . 31
3.3 Efficient implementations . . . 33
3.3.1 Naive implementation . . . 33
3.3.2 Common improvements . . . 34
3.3.3 More improvements . . . 35
3.4 Computational study . . . 37
3.4.1 Results for instances in Class(1) . . . 38
3.4.2 Comparison with previous works . . . 39
3.4.3 Results for instances in Classes (2) and (3) . . . 39
3.4.4 Computation time of one iteration . . . 46
3.5 Computing an optimal branch-decomposition . . . 46
3.5.1 Seymour and Thomas algorithm: Edge Contraction . . . 46
3.5.2 Gu and Tamaki algorithm . . . 48
3.6 Computing optimal branch-decomposition: Efficient Implementation . . . 49
3.7 Computational study . . . 50
3.7.1 Computational study: Edge contraction . . . 50

4 DOMINATING SET Problem in planar graphs 56
4.1 Fomin and Thilikos algorithm 56
4.2 Data reduction 60
4.3 Computational results 64

5 CONNECTED DOMINATING SET problem in planar graphs 72
5.1 Algorithm for planar CDS problem 72
5.1.1 Sphere-cut branch-decomposition based approach 74
5.1.2 Algorithm description 76
5.1.3 Index method 79
5.1.4 Fast matrix multiplication and distance product 84
5.2 Fixed parameter time complexity and bidimensionality 86
5.2.1 Fixed parameter time complexity 86
5.2.2 Bidimensionality 92
5.3 Computational results 94

6 Polynomial Time Approximation Schemes(PTAS) for planar graphs problems 97
6.1 Related methods 97
6.1.1 Planar separator method 98
6.1.2 Outer-planar decomposition method 100
6.2 PTAS for the PLANAR DOMINATING SET problem 102
6.2.1 Algorithm 102
6.2.2 Approximation ratio 104
6.3 Computational study of PTAS for the PLANAR DOMINATING SET problem 106

7 Practical performances of the PLANAR DOMINATING SET algorithms 109
7.1 Heuristic methods 109
7.2 Computational study of PTAS and heuristic methods 110

8 Conclusion and future works 114
8.1 Conclusion 114
8.2 Future works 116

Bibliography 118

Appendix A Branchwidth of random maximal graphs **126**

Appendix B The pre-processing for fast matrix multiplication **128**

B.1 Distance product and DPBF Algorithm 129

B.2 The pre-processing for fast matrix multiplication 131

B.2.1 Phase 1 131

B.2.2 Phase 2 132

B.2.3 Phase 3 134

Appendix C An upperbound for the size of CDS of a partially triangulated $(r \times r)$ -grid **135**

List of Tables

3.1 Computation time (in seconds) of Naive and efficient implementations for Class (1) instances. 40
3.2 Memory usage (in megabytes) of Naive and efficient implementations for Class (1) instances. 41
3.3 Computation time (in seconds) of rat, comprat, and memrat quoted from Table 1 of [68] 42
3.4 Computation time and memory of intersection graphs of segments generated by LEDA. 43
3.5 Computation time and memory of random instances generated by PIGALE. 44
3.6 Computation time (in seconds) of several implementations for k close to the carvingwidth 2bw. 45
3.7 Number of calls for ST Procedure and computation time (in seconds) for Round-Robin method. 51
3.8 Number of calls for ST Procedure and computation time (in seconds) for Random method 51
3.9 Number of calls for ST Procedure and computation time (in seconds) for Greedy method. 52
3.10 Number of calls for ST Procedure when the path simplification is applied for the instances of Class (3). 53
3.11 Number of calls for ST Procedure and computation time (in seconds) for branch-decompositions. 55

4.1 Computational results (time in seconds) of FT Algorithm with the index method in Step III for instances of Classes (1)(6). For the instances marked with *, the time is for computing the dominating number only because the 3 GByte memory is not enough for computing a minimum dominating set. . . . 68
4.2 Memory space (in MBytes) of FT Algorithm with the index method in Step III for instances of Classes (1)(6). For the instances marked with *, the memory space is for computing the dominating number only. X indicates that the problem can not be solved for the instance because it requires more than 3 Gbytes memory space. 69
4.3 Average performance of FT Algorithm over three graphs for each instance size. The time is in seconds and memory is in Mbytes. 69
4.4 The results (time in seconds) of using new data reduction rules and without using the new rules in Step I. 70
4.5 The results (time in seconds) of using distance product method and index method in Step III. 71

5.1 The list of pairs that adding them to some colorings in $B(1)$ generates forbidding components 83
5.2 Computational results (time in seconds) of DPBF Algorithm. For the instances marked with "*", the 3GByte memory is not enough for computing a minimum connected dominating set. 96

6.1 Computational results (time in seconds) of PTAS for the PLANAR DOMINATING SET problem. 108

7.1 Computational results for heuristic methods and PTAS for the PDS problem (time in second). 111
7.2 Computational results for Exact, Greedy and PTAS algorithms for small instances (time in second). 112
7.3 Computational results for Greedy and PTAS for large instances (time in second). 113

B.1 The rules to compute $p(M_{i+1})$ from $p(M_i)$ 133

List of Figures

2.1 Plane graph G is given in dashed lines and the planar dual graph G^* is in solid lines 14
2.2 A planar graph G with 7 vertices and a branch-decomposition of G with width 4 15
2.3 A planar graph G with 7 vertices and its corresponding medial graph 16

3.1 A planar graph with 12 nodes and resulting graph after contracting edge between vertices 3 and 4. 47

4.1 An example of N_1, N_2, N_3 of node v in the left side graph 60
4.2 An example of N_1, N_2, N_3 of nodes v, w in the left side graph 62

5.1 Assigning the proper black colors to $\partial(A_e)$ 78
5.2 An example of clockwise and counter-clockwise order of vertices. 82
5.3 18×10 cylinder $C_{18,10}$ 87
5.4 Supplement graph of $C_{18,10}$ 87
5.5 Graph H is a subgraph of $G_{20,18}$ and corresponding top and bottom faces are colored by grey. 88

6.1 A separator of size of 5 for a planar graph with 20 nodes 99
6.2 A 3-layer outer planar graph 100
6.3 A counter example such that $|D_{(0,2)}| \leq |D'_{(0,2)}|$. 105
6.4 a minimum general dominating set for (k+2)-outer planar subgraph $G_{(0,2)}$ . . 106

A.1 Merge the rooted binary trees T_i into one binary tree. 127

C.1 A general dominating set of R_i (A subgraph of a partially triangulated 20×20-grid graph) 136

C.2 D' is a general dominating set of a partially triangulated 20×20-grid graph . 137

Chapter 1

Introduction

An important research area in graph theory and networks is domination which has been energetically investigated for many years due to its large number of real-world applications. Haynes et al. in their book on domination [67] listed more than 1200 papers in this area. Let $G(V, E)$ be an undirected graph with the set of vertices V and the set of edges E. The classical k-dominating set D of graph $G(V, E)$ is a subset of V containing k vertices, such that for every vertex v in V, either $v \in D$ or v has a neighbour in D. The minimum integer k for which G has a k-dominating set is called the dominating number of G and is denoted by $\gamma(G)$.

The DOMINATING SET problem is to decide that given a graph $G(V, E)$ and an integer k, whether $\gamma(G) \leq k$. The optimization version of this problem is to find the minimum dominating set. This problem has a wide rang of real-world applications. One of the most widely noted applications of this problem is the resource allocation problem. The first instance of the resource allocation problem was introduced by Berge in [13] as the problem of locating minimum number of radars to cover an area. Liu in [83] introduced a similar problem for distributing minimum number of transmitting stations to connect a set of cities. Here the communication links are set up between cities and transmission stations must be located in some of these cities so that every city can receive messages from at least one of these stations through the links. The problem of finding minimum number of cities to set up stations is exactly that of finding the minimum dominating set of the graph having the cities as vertices and the communication links as edges. He also showed that if the induced graph has no isolated vertex, it is possible to set up two groups of transmission stations such that no city has two transmission stations.

Another application of DOMINATING SET problem arises in voting [91]. Assume that a committee needs to be chosen from the staff of a research organization. It is hoped that the committee would best represent the needs and viewpoints of the entire staff but still not too large. To choose the committee every staff member indicates on his ballot his best candidate. Instead of choosing the members with the largest numbers of votes, a structural analysis is made of the votes. A graph of election is constructed, such that vertex u is connected to v if person u voted for person v. Minimum dominating set of this graph is the best group of members that represents the needs and the viewpoints of all the members. The power domination in electronic power networks [66] is a recent application of DOMINATING SET problem. The electric power companies need to continually monitor the state of variables such as voltage and load for their electronic systems. One method of monitoring these variables is to place phase measurement units (PMUs) at selected locations in the system. Since PMUs are expensive devices, it is desirable to minimize their number while the entire system is monitored. Let $G = (V, E)$ be a graph representing an electric power system, where a vertex represents an electrical node and an edge represents a transmission line joining two electrical nodes. PMUs are placed in the vertices of the graph in a way that the PMU at vertex v can monitor incident edges to v and their end vertices. The problem is placing minimum number of PMUs in a subset of vertices such that all edges and vertices of the graph are monitored. This is a graph theory problem closely related to the well-known vertex covering and domination problems. Hence, this problem is not only of interest in the power system industry but also as a new problem in graph theory.

The origin of the dominating set concept traces back to the 1850's, when the following problem was noticed among chess players in Europe: Determine the minimum number of queens that can be placed on a chessboard so that all squares are either attacked by a queen or are occupied by a queen. A large number of domination problem variations have been introduced over the years. More than 75 variations of domination have been addressed in [67]. Below is a list of some of the fundamental types of the domination problem:

1. VERTEX COVER: A vertex v is said to cover every edge incident to v. A vertex cover is a set S of vertices which covers every edge in E.

2. EDGE DOMINATION: A set S of edges is an edge dominating set, if for every edge $e \in E \setminus S$ there exists an edge $f \in S$, such that e and f have a vertex in common.

3. INDEPENDENT DOMINATION: A dominating set D is an independent dominating

set if no two vertices in D are adjacent.

4. CONNECTED DOMINATION: A dominating set D is a connected dominating set if the subgraph induced by D is a connected subgraph of G.

The DOMINATING SET problem is proved NP-complete [59]. However, due to its practical importance this problem has been the subject of many researches. In this thesis we deeply study two problems: DOMINATING SET problem and CONNECTED DOMINATING SET problem. We begin by introducing these problems in details and then review the existing approaches to solve these problems. Since the most efficient algorithms for these problems are based on the notion of branch-decomposition we also introduce this notion. The formal definitions of these problems are given in Chapter 2.

1.1 DOMINATING SET problem

As for most of the NP-hard problems the approximation algorithms and (exact) fixed-parameter algorithms are two well-studied coping strategies for the DOMINATING SET problem.

In general, a minimization problem P of size n is $\alpha-$approximable ($\alpha \geq 1$) if there is a polynomial time algorithm which gives a solution for any instance of P with solution value at most $\alpha.OPT$, where OPT is the value of the optimal solution for that instance of P. If P is $(1+\epsilon)-$approximable for any fixed $\epsilon > 0$ then P has a polynomial time approximation scheme (i.e., has a PTAS). The DOMINATING SET problem for general graphs is $(1 + \log n)-$approximable [74], however is not approximable within a factor $(1-\epsilon)\ln n$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\log\log n})$ [51]. The DOMINATING SET problem has been widely studied on an important class of graphs, the *planar* graphs. A graph is planar if it can be drawn in the sphere with no crossing edges. The DOMINATING SET problem on planer graphs, known as PLANAR DOMINATING SET problem, is known to be NP-hard [59], however, a $(1 + \epsilon)-$approximation algorithms with running time of $2^{O(1/\epsilon)}n^{O(1)}$ has been introduced by Baker in [12] for this problem.

An optimization problem is fixed-parameter tractable, if the value of an optimal solution of the problem, denoted by a parameter k, can be found by an algorithm in polynomial time in the input instance size and in $f(k)$ time where $f(k)$ is a computable function [45]. Such an algorithm for a fixed-parameter tractable problem is called *fixed -parameter*

tractable (FPT) algorithm. Readers may refer to [55] for a survey on new techniques for developing exact exponential time algorithms. It is shown in [45] that for general graphs the DOMINATING SET problem is not fixed-parameter tractable unless some collapses occur between parametrized complexity classes. However PLANAR DOMINATING SET problem is fixed-parameter tractable [45].

A main idea behind many FPT algorithms for the hard problems in graphs is to partition the input graph into subgraphs, find solutions for each subgraph and combine them to get the final solution for the original graph. One of the well known methods for developing FPT algorithms on planar graphs is based on branch-decompositions of planar graphs. Informally, a branch-decomposition of a graph G is a collection of vertex-cut sets represented as links of a tree whose leaves are edges of G. The width of a branch-decomposition is the maximum size of a cut set in the collection and the branchwidth of G, denoted by $bw(G)$, is the minimum width of all possible branch-decompositions of G. Another tool to decompose the graph into subgraphs is tree-decomposition that generates subgraphs which are induced by the set of vertices although in branch-decomposition the subgraphs are induced by the sets of edges. A tree/branch-decomposition based algorithm utilizes decompositions to partition the input graph into subgraphs and applies *dynamic programming* on the subgraphs resulted by this decomposing to solve many combinatorial problems on graphs. The running time of a tree/branch-decomposition based algorithm is usually exponential in the width of the decomposition used and polynomial in the size of input instance. Therefore treewidth/branchwidth are considered bounded parameters. The tree/branch-decomposition based approach gives fixed parameter algorithms. The first fixed-parameter algorithms for the PLANAR DOMINATING SET problem on planar graphs with $\gamma(G) = k$ have running time $O(11^k n)$ [45] and $O(8^k n)$ [48], these algorithms are tree-decomposition based algorithms. Most tree-decomposition based algorithms suffer from the large constants which make them impractical. One efficient way to deal with this problem is to use branch-decomposition instead of tree-decomposition. Fomin and Thilikos gave such an algorithm of running time $O(2^{(3\log_4{}^3)bw(G)}k + n^3)$, where $bw(G)$ is the branchwidth of G [56]. They proved $bw(G) \leq 3\sqrt{4.5\gamma(G)}$ resulting in an $O(2^{15.13\sqrt{k}}+n^3)$ time algorithm for the PLANAR DOMINATING SET problem. Recently, several exponential speedups in fixed-parameter algorithms for various problems on several classes of graphs have been achieved. These techniques reduce the running time of a class of FPT algorithms from $2^{O(k)}n^{O(1)}$ to $2^{O(\sqrt{k})}n^{O(1)}$, where k is the parameter, usually the value of the optimal solution of the problem. This

framework builds on the bidimensionality theory and can be applied to extend the subexponential time fixed-parameter algorithms for the class of problems including VERTEX COVER and DOMINATING SET problems in planar graphs to a broader class of graphs excluding a fixed graph as a minor. The work of [52] further makes such extensions possible for the non-local problems represented by the longest path and CONNECTED DOMINATING SET problems. The survey papers of [37, 53] give a summary on the progress of the subexponential fixed-parameter algorithms, the bidimensionality theory and its algorithmic applications. Examples of such subexponential time algorithms include those that solve the PLANAR k-VERTEX COVER problem, the PLANAR k-DOMINATING SET problem, and PLANAR k-LONGEST PATH problem in $O(2^{3.57\sqrt{k}}k+n), O(2^{11.98\sqrt{k}}k+n^3)$ and $O(2^{10.52\sqrt{k}}n+n^3)$ time, respectively [40].

1.2 CONNECTED DOMINATING SET problem

A dominating set, D, of a graph G is a connected dominating set of G if the graph induced by the vertices of D is a connected subgraph of G and the minimum size of a connected dominating set of G is called *connected dominating number of* G and denoted by $\gamma_c(G)$. The CONNECTED DOMINATING SET (CDS) problem is NP-complete [59]. The best existing approximation ratios for this problem are $2(1+\ln\Delta)$ and $(\ln\Delta+3)$, where Δ is the maximum vertex degree of the input graph [63]. It is proved that the CDS problem is not approximable within a factor of $(1-\epsilon)\ln\Delta$ for any $\epsilon>0$ unless $NP \subseteq DTIME(n^{\log\log n})$[63]. Furthermore, it is fixed-parameter intractable unless the parametrized complexity classes collapse [45, 44].

The PLANAR CDS problem is also NP-complete [59]. However, the PLANAR CDS problem admits a PTAS [36], and is fixed parameter tractable [42, 43]. In many applications in wireless networks, the network nodes have limited power supply [85]. Thus, it requires that the operations at the network nodes to be energy efficient. Planar graphs play a key role in the energy efficient routing and broadcasting. For example, in many wireless network models the Gabriel graph, which is planar, provides a most energy efficient topology for the networks [79]. Practically efficient exact algorithms for the PLANAR CDS problem are of great interests for those applications. Although, the CDS problem and the DOMINATING SET problem looks similar, the existing approaches for the DOMINATING SET problem can not be applied for the CDS problem. The reason is that the connectivity in the CDS problem

is a non-local property (more details in Chapter 5). Informally, in dynamic programming step of the DOMINATING SET algorithm, each partial solution of a subgraph can be identified based on local properties of the subgraph, however in the CDS problem this is not possible which makes the merging partial solutions more difficult. Dorn et al. [42] introduced a new algorithm design technique based on a branch-decomposition of planar graphs with certain geometric properties (known as *sphere-cut decomposition*) and showed that a number of non-local hard problems, including the CDS problem, in a planar graph G can be solved in $O(2^{O(\mathrm{bw}(G))}n^{O(1)})$ time, where $\mathrm{bw}(G)$ is the branchwidth of G. These results imply an $2^{O(\sqrt{k})}$ time fixed-parameter algorithm for the planar CDS problem, where $k = \gamma_c(G)$.

1.3 Basic techniques

As we mentioned in the previous sections the PLANAR DOMINATING SET problem and the PLANAR CDS problem are NP-complete and only approaches to solve these problems are approximation and exact algorithms. The notion branch-decomposition is a powerful tool to develop fixed parameter algorithms for these problem. However the running time of the branch-decomposition based algorithms are exponential in the width of the branch-decomposition used. For graphs with large branchwidth, the branch-decomposition based algorithms are not practical. To approach this issue Baker introduced a new technique, called *outer planar-decomposition*, to decompose the graph into subgraphs with small branchwidth. For every subgraph the problem in practical time and memory can be solved and she showed that combining the solutions of the subgraphs results an approximated solution. Another approach to reduce the running time of a branch-decomposition based algorithm is *kernelization*. In this method for a given graph G, by applying some reduction rules some nodes and edges of G are removed such that the size of optimal solution is not changed. Decreasing the size of the input graph using reduction rules can improve the branch-decomposition based algorithms. In what follows we introduce branch-decomposition based algorithms, outer planar-decompositions and kernelization.

1.3.1 Branch-decomposition of planar graphs

The notation of branch-decomposition is introduced by Seymour and Robertson [98] in proof of the Graph Minors theorem which is known as Wagner's conjecture. Branch-decompositions of graphs and tree-decompositions are closely related. Courcelle [32] and Arnborg et al. [10] showed that many NP-hard problems on graphs can be solved in polynomial time if the tree/branchwidth of the input graph is bounded by a constant. Using tree/branch-decompositions results in some algorithms that have exponential time in the width of tree/branch-decomposition used and polynomial in the size of the input graph. These algorithms are called *tree/branch-decomposition based* algorithms and have two main steps:

1. decomposing the input graph into subgraphs using a tree or branch-decomposition.
2. applying dynamic programming to compute partial solutions for the subgraphs and combining these solutions to find an optimal solution for the original graph.

There is a no big difference between tree-decomposition based algorithms and branch-decomposition based algorithms in theory. But branch-decomposition based algorithms are easier to implement and more practical [56]. In this thesis we concentrate on branch-decomposition based algorithms and for more information on tree-decomposition based algorithms you may refer to [70, 103]. Since every branch-decomposition based algorithm has exponential running time in the width of the branch-decomposition used , finding a branch-decomposition of small width in Step (1) is very important. However it is NP-complete to decide the minimum branchwidth of a general graph [102]. If the problem is restricted to planar graphs, Seymour and Thomas give a decision algorithm (called ST Procedure for short in what follows) which decides if a given planar graph G has a branchwidth at least k in $O(n^2)$ time [102]. Using ST Procedure as a subroutine, they also give an algorithm which constructs an optimal branch-decomposition of G in $O(n^4)$ time [102]. Gu and Tamaki [61] give an improved algorithm which calls ST Procedure $O(n)$ times and runs in $O(n^3)$ time to construct the branch-decomposition.

1.3.2 Outer planar-decomposition

The existing theoretical proof and also our computational results for the DOMINATING SET problem show that the branch-decomposition based algorithms on planar graphs with

large branchwidth are not practical. To approach this issue Baker introduced outer planar-decomposition method to find an approximated solution for solving problems on planar graphs with large branchwidth [12]. In this approach the input graph is embedded on a sphere such that the vertices of the graph are placed on k nested circles which are called layers. It is shown [12] that every subgraph induced by m layers has branchwidth at most $2m$. In outer planar-decomposition method the input graph is divided into subgraphs of constant layers, and for every subgraph a branch-decomposition based algorithm can be applied to generate an optimal solution for the subgraph. Combining these optimal solutions for subgraphs gives an approximated solution for the original graph.

1.3.3 Kernelization

Two encouraging facts about the branch-decomposition based algorithms for PLANAR DOMINATING SET problem are the existence of polynomial time algorithms for computing an optimal branch-decomposition of planar graphs [61, 102] and a linear size kernel for the problem [7]. In particular, Alber et al. [7] gave an $O(n^3)$ time algorithm that for a given planar graph G with $\gamma(G) = k$, produces a reduced graph H (kernel) such that H has $O(k)$ vertices and $\gamma(H) = k' \leq k$. They showed that a minimum dominating set of G can be constructed from a minimum dominating set of H in linear time. These results motivate us to perform a collection of computational studies on PLANAR DOMINATING SET problem [87]. It is also known that the PLANAR CDS problem admits a linear size kernel [72, 86] and such a kernel can be computed in $O(n^3)$ time [72].

1.4 Motivation and contribution

Although, significant theoretical progress has been made towards the fixed-parameter algorithms for the DOMINATING SET problem, limited work has been done on the practical performances of these algorithms and their variations. Alber and et. al. [5, 4] presented the experimental evaluations for the tree-decomposition based algorithms for several problems including the VERTEX COVER and DOMINATING SET problems in planar graphs. A recent experimental study on the heuristic algorithms for the DOMINATING SET problem also been reported in [99]. Major barriers include computing a branch-decomposition in practice and implement the FPT algorithms. Our contribution in this dissertation are four folds:

1. Develop efficient and practical algorithms to compute an optimal branch-decomposition of planar graphs. This provides tools for the evaluation of practical performance of branch-decomposition based algorithms for NP-hard problems in planar graphs (Joint work with Zhengbing Bian).

2. Computational study of an FPT algorithm for the PLANAR DOMINATING SET problem.

3. Computational study of an FPT algorithm for the PLANAR CDS problem that briefly introduced in [42] (we call it DPBF algorithm). In this study we also prove a better upper bound on the running time of DPBF Algorithm, and give an efficient implementation.

4. Develop and implement of a PTAS based on outer planar-decomposition method for the PLANAR DOMINATING SET problem. We show that the mentioned approximation ratio for this problem in [12] is not correct and we prove the correct approximation ratio.

In contribution (1), we develop efficient and practical algorithms to compute the optimal branch-decomposition of a planar graph. The results from previous studies show that the memory usage is a bottleneck for ST Procedure [68]. We introduce several improvements to reduce the memory usage and make it practical. This makes the evaluation of practical performance of FPT algorithms possible. We also study the performance of $O(n^3)$ time algorithm by Gu and Tamaki for computing the optimal branch-decompositions. This contribution is a joint work with Zhengbing Bian, Qianping Gu, Hisaco Tamaki and Yumi Yoshitake that appeared in Proc. of the 10th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX'08)[18] and a part of work is also appeared in Bian's PhD thesis [16].

In contribution (2), we give an efficient implementation for Fomin and Thilikos algorithm [56] for solving the PLANAR DOMINATING SET problem that uses the branch-decomposition based approach. The computational results show that the algorithm can solve the DOMINATING SET problem of large planar graphs in a practical time and memory space for the class of graphs with small branchwidth. For the class of graphs with large branchwidth, the size of instances that can be solved by the algorithm in practice is limited to about one thousand edges due to memory space bottleneck. The practical performances

of the algorithm coincide with the theoretical analysis of the algorithm. Our results suggest that the branch-decomposition based algorithms can become practical for some applications on planar graphs. We also slightly improve the size of kernel for this problem and we implement some heuristic algorithms for the DOMINATING SET problem. We compare their performance with Fomin and Thilikos algorithm.

In contribution (3), we perform computational studies for a non-local problem, the PLANAR CONNECTED DOMINATING SET problem. The work of Dorn et al. [42, 43] suggests a theoretically efficient exact algorithm (called DPBF Algorithm) for the CDS problem in planar graphs of small branchwidth. However, the practical performance of the algorithm is yet to be evaluated. Due to the non-local property of the CDS problem, it is more difficult to implement the tree/branch-decomposition based algorithm for the problem. This may be a hurdle for the computational study of DPBF Algorithm.

Since the DPBF Algorithm is only briefly introduced and the analysis is not explicitly given in the literature [42, 43, 40], we give a detailed description of the algorithm and analyze its running time. By a better analysis, we prove that DPBF Algorithm has running time in $O(2^{4.62bw(G)}\gamma_c(G) + n^3)$, which improves the running time of $O(2^{4.67bw(G)}\gamma_c(G) + n^3)$, implicitly mentioned in [42, 43, 40]. We also introduce a new upper-bound for $bw(G)$ based on $\gamma_c(G)$. It is known that $\text{bw}(G) \leq 3\sqrt{4.5\gamma(G)}$ [57, 56]. Since $\gamma(G) \leq \gamma_c(G)$, DPBF Algorithm solves the PLANAR CDS problem in $O(2^{23.7\sqrt{\gamma_c(G)}}\gamma_c(G) + n^3)$ time. We prove that $\text{bw}(G) \leq 2\sqrt{10\gamma_c(G)} + 32$ for a planar graph G and improve the previous upper bound of $\text{bw}(G) \leq 3\sqrt{4.5\gamma(G)}$. From this result, the PLANAR CDS problem admits an $O(2^{23.54\sqrt{\gamma_c(G)}}\gamma_c(G)+n^3)$ time fixed-parameter algorithm. We also perform a computational study to evaluate DPBF Algorithm on several classes of planar graphs. These classes cover a wide range of planar graphs that have been used in previous computational studies [3, 6, 87].

In contribution (4), we implement a PTAS algorithm based on outer planar-decomposition method for PLANAR DOMINATING SET problem. We use Fomin and Thilikos algorithm to compute the partial solutions for the subgraphs. Baker mentioned that the approximation ratio for the PLANAR DOMINATING SET problem is $\frac{(k+1)}{k}$. However, we give a counter example to show this proof is not correct for the PLANAR DOMINATING SET problem. We show that the approximation ratio for the outer planar-decomposition method is actually $\frac{(k+2)}{k}$. Our introduced PTAS algorithm can be used to compute a dominating set of planar graphs of large branchwidth.

To complete our computational study, we implement some heuristic algorithms for the DOMINATING SET problem and compare their performance with the fixed-parameter algorithm and PTAS algorithms. Heuristic algorithms are faster than the fixed-parameter algorithms but the difference between the results are sometimes considerable.

Our computational study gives a concrete example on using the branch-decomposition based algorithms for solving important local and non-local problems in planar graphs and shows that the PLANAR DOMINATING SET and CDS problems can be solved in practice for a wide range of graphs. This work provides a tool for computing the optimal connected dominating set of planar graphs and may bring the sphere-cut decomposition and noncrossing partitions based approach closer to practice.

1.5 Organization

This thesis is organized as follows: In Chapter 2 we introduce preliminaries and previous works on DOMINATING SET, CONNECTED DOMINATING SET and branch-decompositions. Chapter 3 introduces algorithms for computing branch-decomposition and our algorithms for branch-decomposition of planar graphs. In Chapter 4 we discuss the computational study of Fomin and Thilikos Algorithm for PLANAR DOMINATING SET problem. In Chapter 5 we introduce the details of a sphere-cut decomposition based algorithm (DPBF Algorithm) for solving the PLANAR CDS problem and show the running time of the algorithm and the results of the computational studies of this algorithm. In Chapter 6 we introduce a PTAS algorithm for the PLANAR DOMINATING SET problem and study its performance. In Chapter 7 we study the performance of proposed heuristic algorithms and compare them with the fixed-parameter algorithm for the PLANAR DOMINATING SET problem. Finally, we conclude the thesis in Chapter 8.

Chapter 2

Preliminaries and related works

2.1 Preliminaries

In this chapter, we introduce preliminaries and previous works on branch-decomposition, DOMINATING SET and CONNECTED DOMINATING SET problems on planar graphs.

2.1.1 Basic definition

We use the standard notation of graphs which is used in many graph theory books, such as [109]. Graphs discussed in this thesis are undirected unless otherwise stated. A graph G 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. In this definition, we do not allow self-loop edge but allow parallel edges in G.

For a set $A \subseteq E(G)$ of edges let $V(A) = \cup_{e \in A} e$ be the set of vertices in edges of A. We say a vertex v and an edge e are incident to each other if $v \in e$. We say that two edges e_1 and e_2 are incident to each other if $e_1 \cap e_2 \neq \emptyset$. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a subset $A \subseteq E(G)$ ($U \subseteq V(G)$), we denote by $G[A]$ ($G[U]$) the subgraph of G induced by A (U). For each $X \subseteq V(G)$, we denote by $\delta_G(X)$ the set of edges incident with a vertex in X and a vertex in $V(G) \setminus X$.

Let Σ be a fixed sphere. A set P of points in Σ is a *topological segment* of Σ if it is homeomorphic to an open segment $\{(x, 0) | 0 < x < 1\}$ in Σ. For a topological segment P, we denote by $\overline{P}$ the closure of P and $\mathrm{bd}(P) = \overline{P} \setminus P$ the two *end points* of P. A planar embedding of a graph G is a mapping $\rho : V(G) \cup E(G) \to \Sigma \cup 2^{\Sigma}$ satisfying the following

properties.

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

A graph is planar if it has a planar embedding. A plane graph is a pair (G, ρ), where ρ is a planar embedding of G. We may simply use G to denote the plane graph (G, ρ), leaving the embedding ρ implicit. We do not distinguish a vertex v (resp. an edge e) from its embedding $\rho(v)$ (resp. $\rho(e)$) when there is no confusion. For a plane graph G, each connected component of $\Sigma \setminus (\cup_{e \in E(G)} \overline{\rho(e)})$ is a face of G. We denote by $R(G)$ the set of faces of G. A face $r \in R(G)$ and an edge $e \in E(G)$ are incident to each other if e is a boundary of r in the embedding. Notice that an edge e is incident to exactly two faces. For a face $r \in R(G)$, a vertex v is incident to r if v is an end vertex of an edge incident to r. For a face $r \in R(G)$, let $V(r)$ and $E(r)$ be the sets of vertices and edges incident to r, respectively. For a vertex $v \in V(G)$, let $E(v)$ be the set of edges incident to v.

For a planar graph G, the planar dual G^* of G is defined as that for each face $r \in R(G)$, there is a unique vertex $v_r^* \in V(G^*)$, and for every vertex $v \in V(G)$, there is a unique face $r_v^* \in R(G^*)$. For every edge $e \in E(G)$ incident to faces r and r', there is a unique edge $e^* = \{v_r^*, v_{r'}^*\} \in E(G^*)$ which crosses e. Figure 2.1 gives a plane graph G and its planar dual G^*.

A walk in a graph G is a sequence of edges $e_1, e_2, ..., e_k$ of G, where $e_i = \{v_{i-1}, v_i\}$ for $1 \leq i \leq k$. A walk is closed if $v_0 = v_k$. The length of a walk is the number of edges in the walk. For two vertices u and v in a graph G, the distance $d(u, v)$ is the minimum length of all walks between u and v. The walk with distance $d(u, v)$ is a shortest path between u and v.

For a graph G and a subset $A \subseteq E(G)$ of edges, we denote $E(G) \setminus A$ by $\overline{A}$ when G is clear from the context. A separation of graph G is a pair $(A, \overline{A})$ of subsets of $E(G)$. For each $A \subseteq E(G)$, we denote by $\partial(A)$ the vertex set $V(A) \cap V(\overline{A})$. The *order* of separation $(A, \overline{A})$ is $|\partial(A)| = |\partial(\overline{A})|$.

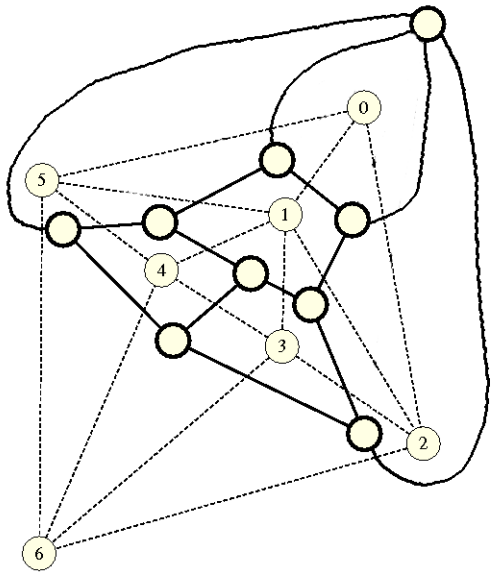
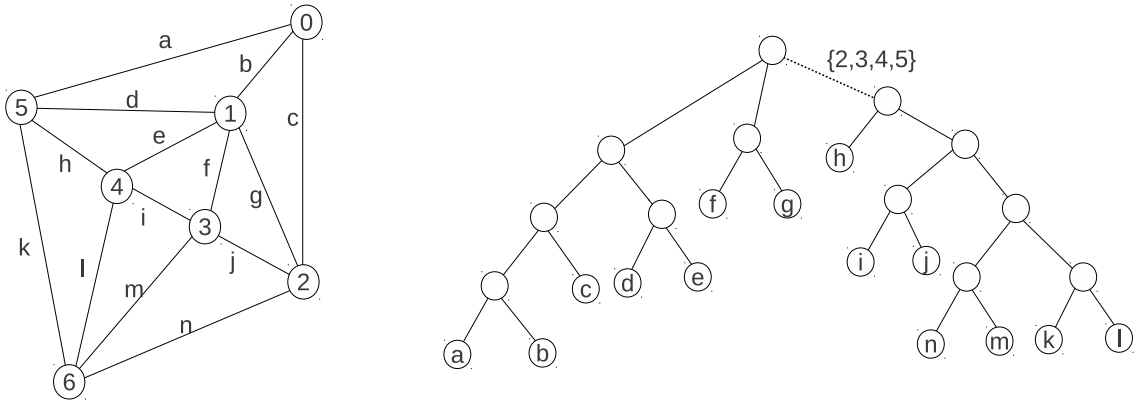


Figure 2.1: Plane graph G is given in dashed lines and the planar dual graph G^* is in solid lines

2.1.2 Branch-decompositions and grid-minors

The notions of branchwidth and branch-decomposition are introduced by Robertson and Seymour [98]. A *branch-decomposition* of graph G is a pair (ϕ, T) where T is a tree, each internal node of which has degree 3 and ϕ is a bijection from the set of leaves of T to $E(G)$. Consider a link e of T and let L_1 and L_2 denote the sets of leaves of T in the two respective subtrees of T obtained by removing e. We say that the separation $(\phi(L_1), \phi(L_2))$ is induced by link e of T. We define the width of the branch-decomposition (ϕ, T) to be the largest order of the separations induced by links of T. The *branchwidth* of G, denoted by $\mathrm{bw}(G)$, is the minimum width of all branch-decompositions of G. In the rest of this thesis, we identify a branch-decomposition (ϕ, T) with the tree T, leaving the bijection implicit and regarding each leaf of T as an edge of G. Figure 2.2 shows a planar graph with 7 vertices, and a branch-decomposition of G with width 4. In this example, the width of the branch-decomposition is corresponding to the order of the separation induced by edge e which is denoted by dash lines in Figure 2.2.

Seymour and Thomas [102] give an algorithm which, given a planar graph G and integer



(a) A planar graph G (b) A branch-decomposition of G

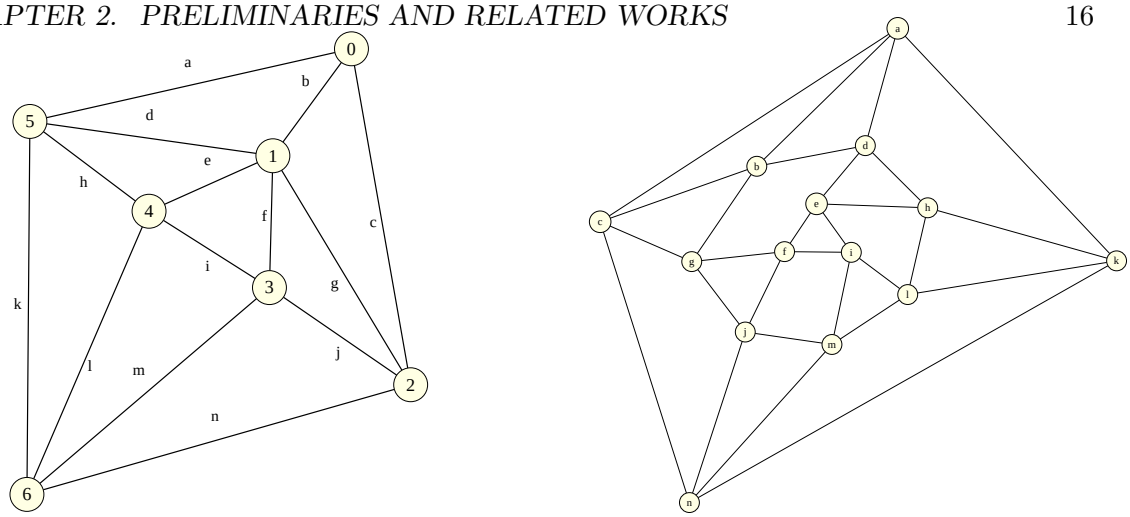
Figure 2.2: A planar graph G with 7 vertices and a branch-decomposition of G with width 4

k decides if $bw(G) \leq k$ in $O(n^2)$. The algorithm is known as the *rat-catching*, also called ST Procedure in this thesis. ST procedure actually works on another decomposition called *carving-decomposition*.

A carving decomposition of G is a tree T_C such that the set of leaves of T_C is $V(G)$ and each internal node of T_C has node degree 3. For each link e of T_C, removing e separates T_C into two subtrees and the two sets of the leaves of the subtrees are denoted by V' and V''. The width of e is the number of edges of G incident to both a vertex in V' and a vertex in V''. The width of T_C is the maximum width of all links of T_C. The carvingwidth of G is the minimum width of all carving decompositions of G.

A more general definition of carving decomposition can be found in [102]. The definition allows positive integer lengths on edges of the graphs. The width of e in T_C for the weighted graph is defined as the sum of weights of edges with an end vertex in V' and an end vertex in V''.

Let G be a plane graph and $R(G)$ be the set of faces of G. The medial graph [102], $M(G)$ of G is a planar graph with an embedding such that $V(M(G)) = \{u_e | e \in E(G)\}, R(M(G)) = \{r_s | s \in R(G)\} \cup \{r_v | v \in V(G)\}$, and there is an edge $\{u_e, u_{e'}\}$ in $E(M(G))$ if the edges e and e' of G are incident to a same vertex v of G and they are consecutive in the clockwise (or counter clockwise) order around v. $M(G)$ in general is a multigraph but has $O(|V(G)|)$



(a) A planar graph G (b) Medial graph of G

Figure 2.3: A planar graph G with 7 vertices and its corresponding medial graph

edges. Figure 2.3 shows a plane graph and its medial graph.

Seymour and Thomas [102] show that the carvingwidth of $M(G)$ is exactly twice the branchwidth of G and an optimal carving-decomposition of $M(G)$ can be translated into an optimal branch-decomposition of G in linear time. To decide whether a planar graph G has the branchwidth at least an integer β, ST Procedure actually decides whether $M(G)$ has the carvingwidth at least 2β.

Let G be a plane graph. We say that a curve μ on the sphere Σ is *G-normal* if μ does not intersect with itself or any edge of G. A *noose* of G is a closed G-normal curve on Σ. Let ν be a noose of G and let R_1 and R_2 be the two open regions of the sphere separated by ν. Then, ν induces a separation $(A, \overline{A})$ of G, with $A = \{e \in E(G) \mid \rho(e) \subseteq R_1\}$ and $\overline{A} = \{e \in E(G) \mid \rho(e) \subseteq R_2\}$. We also say that noose ν induces edge-subset A of G if ν induces a separation $(A, \overline{A})$ having A on one side. We call a separation or an edge-subset *noose-induced* if it is induced by some noose. A branch-decomposition T of G is a *sphere-cut decomposition* if every separation induced by a link of T is noose-induced [42, 43]. It is known that every plane graph G has an optimal branch-decomposition (of width $\mathrm{bw}(G)$) that is a sphere-cut decomposition and such a decomposition can be found in $O(n^3)$ time [102, 61].

2.1.3 Domination problems

A dominating set D of G is a subset of $V(G)$ such that for every vertex $u \in V(G)$, $u \in D$ or u is incident to a vertex $v \in D$. The *dominating number* of G, denoted by $\gamma(G)$, is the minimum size of a dominating set of G. The DOMINATING SET problem is to decide if $\gamma(G) \leq k$ for a given graph G and integer k. The optimization version of the problem is to find a dominating set D with $|D| = \gamma(G)$. The DOMINATING SET problem is a core NP-complete problem in combinatorial optimization and graph theory [59]. The rich literature of algorithms and complexity of DOMINATING SET problem can be found in [67].

A subset D of $V(G)$ is a CONNECTED DOMINATING SET (CDS) of G if D is a dominating set of G and the subgraph $G[D]$ induced by D is connected. The *connected dominating number* of G, denoted by $\gamma_c(G)$, is the minimum size of a CDS of G. The CDS problem is to decide if $\gamma_c(G) \leq k$ for a given graph G and integer k. The optimization version of the CDS problem is to find a minimum CDS of an input graph. The CDS problem is an important variant of the DOMINATING SET problem and has wide practical applications in wireless ad hoc or sensor networks such as virtual backbone construction [29], energy efficient routing and broadcasting [19]. Notice that $\gamma(G) \leq \gamma_c(G) \leq 3\gamma(G) - 2$.

2.2 Previous works

Decompositions play an important role in graph theory. Various decompositions of the graphs such as decomposition by clique separators, tree-decomposition and branch-decomposition are often used to design efficient graph algorithms. There are even interesting general results stating that a variety of NP-complete graph problems can be solved in linear time for graphs of bounded treewidth and bounded cliquewidth [10, 32]. The common approach to design efficient algorithms using decompositions works as follows: smaller graphs are generated recursively until the obtained graph can not be decomposed further. The algorithm solves the problem on these subgraphs and combine the solutions recursively to achieve the solution for the original graph. To achieve an efficient algorithm, the input graph must be restricted to graph classes that are nicely decomposable with the respect to the decomposition method. Several decompositions have been studied in this direction. Modular decomposition is one of the earliest methods to decompose a graph [58] (more information can be found in the thorough survey [89]). The recent researches on modular decompositions show that using

this method results efficient algorithms for solving some NP-complete problems [25, 23, 26]. Tarjan also showed that some NP-complete problems which are solvable using decomposition by clique-separators [105]. Many NP-complete problems can be solved by polynomial time (or even linear time) algorithms using tree-decomposition or clique decompositions if the treewidth or cliquewidth of the graph is bounded [10, 32, 46, 77]. However, the huge constant behind the Big-Oh makes the linear time algorithms impractical.

2.2.1 Branch-decomposition

The notions of *branchwidth* and *branch-decompositions* are introduced by Robertson and Seymour [98] in relation to the more celebrated notions of *treewidth* and *tree-decompositions* [96, 97]. For an arbitrary graph G, there is a linear relation between the treewidth $tw(G)$ and the branchwidth $bw(G)$ of G, $bw(G) \leq tw(G)+1 \leq \lfloor \frac{3bw(G)}{2} \rfloor$ which is proven by simple translation between branch-decomposition and tree-decomposition in [98]. A graph of small branchwidth (or treewidth) admits efficient dynamic programming algorithms for a vast class of problems on the graphs [10, 20]. There are two major steps in a branch/tree-decomposition based algorithm for solving a problem: (1) computing a branch/tree-decomposition with a small width and (2) applying a dynamic programming algorithm based on the decomposition to solve the problem. Step (2) usually runs in exponential time in the width of the branch/tree-decomposition computed in Step (1). So it is extremely important to decide the branchwidth/treewidth and compute the optimal decompositions. It is NP-complete to decide whether the width of a given general graph is at least an integer β if β is part of the input, both for branchwidth [102] and treewidth [9]. When the branchwidth (treewidth) is bounded by a constant, both the branchwidth and the optimal branch-decomposition (treewidth and optimal tree-decomposition) can be computed in linear time [21, 27]. However, the huge constants behind the Big-Oh make the linear time algorithms only theoretically interesting. It is proved that the optimal branchwidth of a planar graph can be computed in polynomial time [102]. This is the reason that the branch-decomposition based algorithms for problems on planar graphs have recently received more attention [42, 56]. Seymour and Thomas in [102] introduce a decision algorithm (called ST Procedure) to decide the optimal branchwidth of a planar graph in $O(n^2)$ time. Using this decision algorithm they showed that the optimal branch-decomposition can be computed in $O(n^4)$ [102]. Gu and Tamaki further optimized this result and showed that using the same decision algorithm the optimal branch-decomposition of a planar graph can be computed in $O(n^3)$ [61]. The computational

study of ST Procedure shows that its straightforward implementation is not practical where the memory usage is the bottleneck [68].

Approximation algorithms for computing the width and minimum-width decompositions of general graphs have been extensively studied as well (see [8, 24] for literature). Because of the relationship between treewidth/tree-decomposition and branchwidth/branch-decomposition stated above, the approximation problems for these two types of decompositions are almost equivalent. For general graphs, the best known approximation ratio computable in polynomial time for the minimum treewidth is $O(\sqrt{\log k})$, where k is the optimal width [47], and constant-factor approximation algorithms take time exponential in the optimal width [8, 102]. For planar graphs, the best-known approximation result for treewidth is the obvious $O(n^2 \log n)$ time 1.5-approximation algorithm, which uses the rat-catching algorithm of Seymour and Thomas [104] and a binary search. Tree-decompositions take $O(n^3)$ time for 1.5-approximation. Bodlaender, Grigoriev and Koster give another constant-factor approximation algorithm for the treewidth of planar graphs that runs in $O(n^2 logn)$ time but uses less memory [24].

One hurdle for applying branch/tree-decomposition based algorithms in practice is the difficulty of computing a good branch/tree-decomposition because of the NP-hardness and huge hidden constants problems. Recently, the branch-decomposition based algorithms with practical importance for problems in planar graphs have been receiving increasing attention [42, 56]. This is motivated by the fact that an optimal branch-decomposition of a planar graph can be computed in polynomial time by Seymour and Thomas algorithm [102] and the algorithm is reported efficient in practice [68, 69]. Notice that it is open whether computing the treewidth of a planar graph is NP-hard or not. The result of the branchwidth implies a 1.5-approximation algorithm for the treewidth of planar graphs. Readers may refer to the recent papers by Bodlaender [22] and Hicks et al. [70] for extensive literature in the theory and application of branch/tree-decompositions.

Hicks proposes a divide and conquer heuristic algorithm to reduce the number of calls for ST Procedure [69]. All known algorithms for computing the optimal branch-decomposition of a planar graph rely on ST Procedure and thus an efficient implementation of the procedure plays a key role in computing the branch-decompositions. A straightforward implementation of ST Procedure requires $O(n^2)$ bytes of memory. Because the constant behind the Big-Oh is non-trivial, the memory required by the straightforward implementation is reported in [68] a bottleneck for solving large instances with more than 5,000 edges. Hicks proposes

memory friendly implementations in the cost of performing re-calculations and increasing the running time of ST Procedure to $O(n^3)$ [68].

2.2.2 DOMINATING SET problem

The DOMINATING SET problem is NP-complete [59]. For an arbitrary undirected graph G of n vertices, the DOMINATING SET problem is known $(1+\log n)-$approximable [74], but not approximable within a factor of $(1-\epsilon)\ln n$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\log\log n})$ [51]. Two 2-approximation algorithms have been proposed for other variations of DOMINATING SET problem, the VERTEX COVER [100] and the EDGE DOMINATION [110] problems. Karakostas in [76] proposed a $(2-\theta(1/\sqrt{\log n}))-$approximation algorithm for the VERTEX COVER problem and Chlebk and Chlebkov [30] proved that it is NP-hard to approximate EDGE DOMINATION problem within any factor better than 7/6. Furthermore, it has been proved that the INDEPENDENT DOMINATING SET problem can not be approximated within $n^{1-\epsilon}$ unless $P = NP$ [65]. The outer planar-decomposition method for solving the PLANAR DOMINATING SET problem [12] can be used to drive PTAS for other variations of dominating set on the planar graphs such as PLANAR VERTEX COVER, PLANAR INDEPENDENT DOMINATION. For the DOMINATING SET problem the only remaining hope is to design exact algorithms with good exponential running times. The best known exponential time algorithm for DOMINATING SET problem is introduced in [106] with time $O(1.5063^n)$. Two exponential time algorithms have also been introduced for INDEPENDENT DOMINATION SET problem with time complexity $O((\sqrt{3})^n)$ [84] and EDGE DOMINATING SET problem with time complexity $O(1.3226^n)$ [107].

The DOMINATING SET problem is also known fixed-parameter intractable unless the parametrized complexity classes collapse [45, 44]. If the problem is restricted to planar graphs, it is known as the PLANAR DOMINATING SET problem which is still NP-hard [59]. But the PLANAR DOMINATING SET problem is known admitting a PTAS [12] and fixed-parameter tractable [45].

The PLANAR DOMINATING SET problem has been extensively studied. The first fixed-parameter algorithm for PLANAR DOMINATING SET problem with running time $O(11^k n)$ where $k = \gamma(G)$ was proposed in [45]. This running time was improved to $O(8^k n)$ in [48]. The exponential speedups give algorithms for the problem with running time $O(2^{c\sqrt{k}}n)$,

where c is some constant [50, 56, 75]. Most of the subexponential algorithms use a tree-decomposition based algorithm: For a planar graph G with $\gamma(G) = k$, a tree-decomposition of width $b\sqrt{k}, b$ is a constant, is computed and the dynamic programming part runs in $O(2^{2b\sqrt{k}}n)$ time [50]. One problem with those algorithms is that the constant $c = 2b$ is too large for solving the PLANAR DOMINATING SET problem in practice. Instead of a tree-decomposition, a branch-decomposition can be used in the above dynamic programming algorithms for the PLANAR DOMINATING SET problem. Fomin and Thilikos give such an algorithm (called FT Algorithm in what follows) of running time $O(2^{(3\log_4^3)\mathrm{bw}(G)}k + n^3)$, where $\mathrm{bw}(G)$ is the branchwidth of G [56]. Fomin and Thilikos prove that $\mathrm{bw}(G) \leq 3\sqrt{4.5k}$ and $O(2^{(3\log_4^3)bw(G)}) = O(2^{15.13\sqrt{k}})$, reducing the constant c to 15.13 [56, 57]. Dorn proposes an approach of applying the distance product of matrices to the dynamic programming step in branch/tree-decomposition based algorithms for the problem [40]. If a conventional $O(n^3)$ time algorithm is used for the distance product of matrices, this approach has the same constant $c = 15.13$ as that of FT Algorithm. It is known that the distance product of integer matrices can be realized by the fast matrix multiplication [111]. If the distance product of matrices is realized by the $O(n^\omega)(\omega < 2.376)$ time fast matrix multiplication method [31], the constant c is improved to 11.98. However, the constant hidden in the Big-Oh may be huge when the fast matrix multiplication is used. Dorn also proves that the PLANAR DOMINATING SET problem can be solved in $O(3^{tw(G)}n^{O(1)})$ time, where $tw(G)$ is the treewidth of G [41]. The tree-decomposition used in Dorn's proof is closely related to the branch-decomposition and the algorithm of [41] has the same sublinear exponent in the time complexity as that of the algorithm in [40]. An encouraging fact on branch-decomposition is that an optimal branch-decomposition of a planar graph can be computed in polynomial time [61, 102]. This makes the branch-decomposition based algorithms receiving increasing attention for the problems on planar graphs.

Another important progress on the algorithmic tractability of the PLANAR DOMINATING SET problem is that the problem is shown having a linear size kernel [49]. Kernelization is an effective approach to speed up fixed-parameter tractable algorithms. Let P be a fixed-parameter tractable problem characterized by a parameter k, the approach is before starting a cost-intensive exact algorithm doing a polynomial time pre-processing phase to shrink the input data of size n to a smaller instance. The solution for the original input then can be reconstructed in polynomial time in n using a solution for the shrunk instance. It is then

hoped that the size of the problem kernel is upper-bounded by a polynomial in k, independent of n. More specifically, for the PLANAR DOMINATING SET problem, Alber et al. give an $O(n^3)$ time algorithm which, given a planar graph G with $\gamma(G) = k$, produces a reduced graph H (kernel) such that H has $O(k)$ vertices, $\gamma(H) = k' \leq k$, and a minimum dominating set of G can be constructed from a minimum dominating set of H in linear time [49]. In general, H and k' are smaller than G and k, respectively, since in the reduction process, a number of vertices in a minimum dominating set of H have been decided. This reduction process reduces the sub-linear exponent from $c\sqrt{k}$ to $c\sqrt{k'}$ and thus improves the running time of the fixed-parameter algorithms for the PLANAR DOMINATING SET problem.

2.2.3 CONNECTED DOMINATING SET (CDS) problem

The CDS problem is known to be NP-complete [59]. Guha and Khuller in [63] proposed two approximation algorithms for the CDS problem on an arbitrary graph with approximation ratio $2(1 + ln(\Delta))$ and $(ln(\Delta) + 3)$ respectively, where Δ is the maximum node degree of the input graph. The first algorithm employs a greedy approach for solving the CDS problem as follows. It starts by an empty set D and iteratively adds vertices to the D. In each iteration the algorithm includes a vertex or a pair of vertices with maximum number of neighbours which are not dominated yet to the D. The second algorithm computes a connected dominating set in two steps. In the first step it finds a dominating set of the input graph, in the second step it includes some vertices to make the dominating set connected.

They also proved that getting an approximation ratio better than $(1 - \epsilon)\ln(\Delta)$ for any fixed $\epsilon > 0$ is not possible unless $NP \subseteq DTIME[n^{O(loglogn)}]$ [63]. The first exact (exponential) algorithm for the CDS problem with running time $O(1.9407^n)$ was proposed by Fomin et al. in [54]. This algorithm is based on "stay connected" strategy which means that all partial solutions generated recursively must be connected.

The CDS problem remains NP-hard even when restricted to the class of planar graphs [59]. The CDS problem on general graphs is not fixed parameter tractable unless fixed parameter complexity hierarchy collapses and admits a sub-exponential time parametrized algorithm for planar graphs [36].

Although the CDS problem and DOMINATING SET problem are closely related, they are rather different from the point of view of exact algorithms. In particular, the techniques used to solve the DOMINATING SET problem do not seem to work for the CDS problem.

One of the main reasons of discrepancy is that *connectivity* is a non-local property: very often exact algorithms for DOMINATING SET, INDEPENDENT SET problems are based on the local structure of the problem, theses algorithms seem not able to capture non-local properties such as connectivity. The CDS problem is an example of non-local problems. The explicit definitions of local and non-local properties are given in Chapter 5.

Dorn et al. in [42] proposed a new frame work (called DPBF ALgorithm in what follows) to design sub-exponential and parametrized algorithms for problems that need non-local information in planar graphs. This framework is based on a combination of the geometric properties of branch-decomposition of planar graphs and applying dynamic programming on planar graphs based on properties of non-crossing partitions (more details can be found in Chapter5).

Based on this framework, sub-exponential algorithms have been proposed for weighted Hamiltonian Cycle and Graph Travelling Saleman problems with time complexity $O(2^{6.903\sqrt{n}})$ and $O(2^{9.8594\sqrt{n}})$ respectively in [42]. It has been mentioned that this approach can be used to design parametrized algorithms as well [42]. For example Dorn et al. introduce the first $O(2^{O(\sqrt{k})}n^{O(1)})$ time algorithm for parametrized Planar k cycle by showing that for a given k we can decide if a planar graph on n vertices has a cycle of length at least k in time $O(2^{13.6\sqrt{k}}n + n^3)$ [42]. They also suggest that the PLANAR CDS problem can be solved in $O(2^{O(\text{bw}(G))}n + n^3)$ and $O(2^{9.822\sqrt{n}}n + n^3)$ time [43][1]. It is mentioned in [40] that the running time can be further improved to $O(2^{8.11\sqrt{n}}n + n^3)$ if the fast distance matrix multiplication is applied to the second step. The time bound $O(2^{O(\text{bw}(G))}n + n^3)$ implies that the PLANAR CDS problem admits an $O(2^{O(\sqrt{\gamma_c(G)})}n + n^3)$ time fixed-parameter algorithm. It is known that the PLANAR CDS problem admits a linear size kernel [72, 86] and such a kernel can be computed in $O(n^3)$ time [72]. Applying the algorithm of [72] to shrink the input graph G into a linear size kernel, the DPBF Algorithm solves the PLANAR CDS problem in $O(2^{O(\sqrt{\gamma_c(G)})}\gamma_c(G) + n^3)$ time.

2.2.4 PTAS and heuristics for DOMINATING SET problem

For an NP-hard problem, there is no polynomial time exact algorithm unless $P = NP$. If an "almost" optimal solution is enough, applying approximation techniques or heuristic algorithms can be used. Readers may refer to [108] and [94, 92] for a survey on approximation

[1]The constant in $O(\text{bw}(G))$ is not explicitly given in [42, 43]

algorithms and heuristic methods respectively.

PTAS

There are many researches to improve approximation ratio of algorithms for NP-hard problems, but in many cases improving the approximation ratio is not possible unless some collapses occur between complexity classes. For example Arora et al. in [11] prove that a class of NP-hard problems including vertex cover, maximum satisfiability, maximum cut, metric TSP, Steiner trees and shortest superstring does not have a PTAS, unless $P = NP$.

Studies on NP-complete problems show that there are many problems which are easier to approximate on planar graphs. For example maximum independent set is inapproximable within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ unless $P = NP$, while for planar graphs there is a 4-approximation algorithm. Another example is the DOMINATING SET problem defined on a general graph with n nodes, this problem is not approximable within $(1-\epsilon)\ln n$ unless $P = NP$, but the PLANAR DOMINATING SET problem has a PTAS.

There are two main general methods to find PTASs for NP-hard problems on planar graphs: *separator method* and *outer-planar decomposition method*.

Separator Method: This method is based on a famous theorem called *Planar Separator Theorem* [81]. Based on this theorem for every planar graph G with n vertices there is a separator of size $O(\sqrt{n})$, whose removal splits the graph into subgraphs of size at most $\frac{2}{3}n$. Finding this separator can be done in polynomial time.

In the separator method the input graph is recursively split to subgraphs util the size of resulting subgraphs is a constant. Since the size of the subgraphs is small the problem on these subgraphs can be solved by a brute-force approach. Then these partial solutions are combined to generate an approximated solution for the original graph. This approach can generate a PTAS only for the problem if the optimal solution is at least some constant factor times n. Some researches such as [90] show that this method is not practical.

Outer-planar decomposition method: In [12] Baker introduces a general method to obtain approximation schemes for various NP-complete problems on planar graphs. This method is based on decomposing a general planar graph into k-outer planar subgraphs. A k-outer planar graph has an embedding with at most k nested disjoint cycles. She shows that the structure of k-outer planar graphs is adaptable with dynamic programming.

Baker shows that for general planar graphs if the problem P is a maximization problem, such as maximum independent set, this technique gives for each k a linear time algorithm

that produces a solution whose size is at least $k/(k+1)$ optimal. If the problem is a minimization problem, such as minimum vertex cover, it gives for each k a linear-time algorithm that produces a solution whose size is at most $(k+1)/k$ optimal. The details of the solution of the maximum independent set problem can be found in [12]. This method resolves two disadvantages of the Separator method. In addition, in [12] some NP-complete problems are listed which are solvable in polynomial time on k-outer planar graphs for constant k.

Heuristic methods

As we mentioned before for many problems on planar graphs using decompositions and applying dynamic programming is effective but its time-complexity often is too high and unacceptable if the branchwidth of the input graph is large. For example the branch-decomposition based algorithm introduced by Fomin and Thilikos is not practical for graphs of large branchwidth. Practically, for many realistic optimization problems heuristics may be attractive in terms of efficiency and solution quality for graphs of large branchwidth.

A heuristic method or heuristic for short is a procedure that determines good or near-optimal solutions for an optimization problem. As opposed to exact methods such as FPT algorithms, heuristics carry no guarantee that an optimal solution will be found. Some heuristics do not guarantee that the found solutions are within a certain ratio of the optimal solution. Many classical heuristics are based on local search procedures, which iteratively move to a better solution (if such solution exists) in a neighbourhood of the current solution. A procedure of this type usually terminates when the first local optimum is obtained. One of the most famous heuristics is *greedy*. The straightforward greedy strategy for finding a small dominating set in a graph consists of choosing vertices which cover the largest possible number of previously uncovered vertices. Parekh has done some theoretical analysis of this algorithm [93]. Several variations of the greedy heuristic for the DOMINATING SET problem are described and their practical performances are reported in [99]. The detailed descriptions of these variations can be found in Chapter 7.

Chapter 3

Branch-decomposition of planar graphs

3.1 Optimal branchwidth of planar graphs

Recently, there have been increased interests in exact algorithms for optimization problems. Many of the exact algorithms use dynamic programing based on tree/branch-decompositions. An optimization problem on a graph with small branchwidth admits efficient dynamic programming algorithm. Therefore, finding a tree/branch-decomposition of small width is a key step in these algorithms. It is NP-complete to decide whether the branchwidth/treewidth of a given general graph is at least an integer β, given that β is part of the input[102, 1]. When the branchwidth or treewidth is bounded by a constant, both branchwidth and treewidth can be computed in linear time [21, 27]. However, the huge constants behind the Big-Oh make the linear time algorithm only theoretically interesting.

Seymour and Thomas showed that an optimal branch-decomposition of a planar graph can be computed in polynomial time [102]. Their algorithm is reported efficient in practice [42, 43].

For a given planar graph G with n vertices and constant β, ST Procedure decides if G has a branchwidth at least β in $O(n^2)$ time [102]. They also propose an algorithm to compute an optimal branch-decomposition of G, that calls ST Procedure $O(n^2)$ times, and runs in $O(n^4)$. Gu and Tamaki improved this result in [61] by proposing an algorithm that calls ST Procedure $O(n)$ times, resulting in an $O(n^3)$ algorithm. In a study by Hicks [68] it is reported

that the straightforward implementation of ST Procedure needs $O(n^2)$ memory which limits the application of ST Procedure. He proposed two memory friendly implementations of ST Procedure with the cost of increasing the running time to $O(n^3)$ [68]. It is not known whether computing an optimal tree-decomposition of a planar graph is NP-hard or not. From the linear relation between treewidth and branchwidth, $bw(G) \leq tw(G) + 1 \leq 1.5bw(G)$, the algorithm of Seymour and Thomas gives a 1.5-approximation algorithm for treewidth of the planar graph.

In this chapter we propose several efficient implementations of ST Procedure and efficient implementations of algorithms for computing the optimal branch-decomposition introduced in [102]. This chapter includes two main parts: in the first part two groups of improvements are introduced for efficient implementation of ST Procedure. Group (1) does not perform re-calculation and runs in $O(n^2)$ time. Using these improvements the optimal branchwidth of a set of instances of size up to one hundred thousand edges can be computed within 500 mega bytes (MB) of memory and a few hours. The improvements in Group(2) are based on the recalculations. The implementations of ST Procedure using this group of improvements can compute an optimal branchwidth of another set of instances with one hundred thousand of edges within 200 MB memory.

The computational results reported in this chapter and in [68] show that the straightforward implementations need at least 1 giga bytes (GB) of memory for instances of size up to 5000 edges. Our most time efficient implementation is faster than the straightforward one by a factor of $3 \sim 15$. We compared the performance of our implementations with those reported in [68] on the same set of instances. However, due to the lack of their software availability we were not able to compare the two methods on similar computational platforms. Thus, our comparison is across two different platforms. However, this difference is largely irrelevant when comparing the memory usage which is the main focus of our implementations. Our most memory efficient implementations of Group (1) and Group (2) use at most 1/4 memory and 1/8 memory, respectively, compared with the previous memory friendly implementations in [68].

In addition, our running time on a machine with 3.06GHz CPU was faster compared with the implementations in [68] by a factor of $100 \sim 400$ and a factor of $100 \sim 200$ for Group (1) and Group (2) most memory efficient implementations, respectively. Notice that the computational platforms used in [68] had 194MHz CPU's.

In the second part of this chapter, we propose several efficient implementations of the

algorithms introduce by Seymour and Thomas (called Edge Contraction (EC) method) and also those introduced by Gu and Tamaki [61] (called GT Algorithm) for computing an optimal branch-decomposition. The EC algorithm is based on contracting the graph edges following a call of ST Procedure for every contraction. In the EC algorithm no method for choosing the contracting edges has been specified. Thus, we examine several different methods including Greedy, Round Robin and Random methods for choosing the contracting edges. Our computational results show that the Round Robin gives the best performance. Another improvement is achieved by simplifying the input graph by replacing the simple paths in the graph with an edge. The results show that depending on the structure of the input graphs, this modification can improve the running time by a factor of $2.92 \sim 16.16$ percent for a class of the instances. The implementation of GT Algorithm suggests that although this algorithm is theoretically faster than the EC algorithm, its performance is almost the same due to the processing of more complex data structures.

The content of this chapter is a joint work with Zhengbing Bian, Qianping Gu, Hisaco Tamaki and Yumi Yoshitake and appeared in Proc. of the 10th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX'08)[18] and some parts of this chapter are also appeared in Bian's PhD thesis [16].

3.1.1 Rat-catching game

In this section we review ST Procedure. Readers may refer to [102] for more details. ST Procedure is often called *rat-catching* algorithm as it can be described by a rat-catching game introduced in [102].

This game is a two-players game, a rat and a rat-catcher. The game is on a plane graph G representing a floor plan of a house. Every face and edge of G is interpreted as a room and a wall of a room, respectively. A vertex is a corner of a room. The rules for the game are as follows.

(R1) The rat-catcher selects a room.

(R2) The rat selects a corner of a room (a vertex of G).

(R3) The rat-catcher selects a room adjacent to the current room and moves to the wall between the two rooms (the edge of G incident to the current face and the selected face). The rat-catcher generates a noise of a fixed level that may make walls noisy. The condition of making a wall noisy will be given later.

(R4) The rat moves to a different corner via walls or stays at the current corner. The rat can not use a noisy wall but can use as many quiet walls as possible in one move.

(R5) The rat-catcher moves to the room it selected and can not change its mind to move back to the previous room. The rat-catcher keeps making noise.

(R6) If the rat is at a corner, all walls incident to the corner are noisy, and the rat-catcher is in a room with this corner then the rat-catcher catches the rat and wins the game. Otherwise go to (R3).

For the planar dual G^* of G, let v_r^* and e^* be the corresponding vertex and edge of G^* to the face r and edge e of G, respectively. Let k be the noise level produced by the rat-catcher. When the rat-catcher is on edge e, edge f is noisy if and only if there is a closed walk of length smaller than k containing edges e^* and f^* in G^*. Similarly, when the rat-catcher is in face r, edge f is noisy if and only if there is a closed walk of length smaller than k containing vertex v_r^* and f^* in G^*. The rat-catcher wins the game if the rat is at a vertex v with node degree smaller than k and the rat-catcher is in a face incident to v. The rat wins the game if there is a scheme by which the rat can escape from the rat-catcher for ever. We use $RC(G,k)$ to denote the rat-catching game on G and k. Seymour and Thomas show that the rat wins the game $RC(G,k)$ if and only if G has carvingwidth at least k and give ST Procedure which, given G and k, computes the outcome of the game $RC(G,k)$ [102].

3.1.2 Seymour and Thomas algorithm

In this section we describe ST Procedure in details. The following presentation is different from the original one which is based on a notation called antipodality [102].

To describe ST Procedure we need to define some notations which are used in ST Procedure. Given G and k, G_e is defined to be the subgraph of G obtained by deleting noisy edges from G when the rat-catcher is on edge e. For every face $r \in R(G)$, let

$$S_r = \{(r,v) | v \in V(G)\}$$

and $S = \bigcup_{r \in R(G)} S_r$. For each $e \in E(G)$, let

$$T_e = \{(e,C) | C \text{ is a connected component of } G_e\}$$

and $T = \bigcup_{e \in E(G)} T_e$.

The game $RC(G,k)$ can be described by a bipartite graph $H(G,k)$, where the vertex set of $H(G,k)$ is $S \cup T$ and there is an edge between $(r,v) \in S$ and $(e,C) \in T$ if face r is incident to edge e and v is a vertex of C. The vertices of $H(G,k)$ can be interpreted as the states of the game and the edges between vertices indicated the possible state transmission of the game. A game state of $RC(G,k)$ is called a *losing state* if the rat will lose the game at that state. ST Procedure deletes the losing states from $H(G,k)$ gradually. The deletion process is reported until no further deletion is possible. If after finishing the deletion process there is an escaping scheme for every face r and every edge e (indicated by non-empty X_r and X_e) the rat wins.

ST Procedure

Input: A non-null connected plane graph G, a planar dual G^* of G, an integer $k \geq 0$.

Output: Decides if G has carvingwidth at least k.

1. If the maximum node degree of G is at least k then output G has carvingwidth at least k and terminate.

2. For each face $r \in R(G)$, let $X_r = S_r$. For each edge $e \in E(G)$, compute G_e and let $X_e = T_e$. For each $(e,C) \in X_e$ and the faces r and r' incident to e, let $c(r,e,C) = |V(C)|$ and $c(r',e,C) = |V(C)|$, where $V(C)$ is the set of vertices of C.

3. For each face r and each state $(r,v) \in X_r$ with $v \in V(r)$, put (r,v) to a stack L and delete (r,v) from X_r.

4. If L is empty then go to the next step. Otherwise, remove a state x from L. Assume that $x = (r,v)$ is a state for a face ($x \in S$). For each edge e incident to r, find the state $(e,C) \in X_e$ such that C contains v. Decrease $c(r,e,C)$ by one. If $c(r,e,C)$ becomes 0 and $(e,C) \in X_e$ then put (e,C) to L and delete (e,C) from X_e. Assume that $x = (e,C)$ is a state for an edge ($x \in T$). If there is a face r incident to e such that $c(r,e,C) > 0$ then for each vertex v of C and $(r,v) \in X_r$ put (r,v) to L and delete (r,v) from X_r. Repeat this step.

5. If X_r is non-empty for every $r \in R(G)$ and X_e is non-empty for every $e \in E(G)$ then output G has carvingwidth at least k, otherwise output G has carvingwidth smaller than k.

To compute G_e for each e, ST Procedure needs to find the quiet edges when the rat-catcher is on edge e. An edge f is quiet and will be included in G_e if every closed walk in

G^* that contains e^* and f^* has length at least k. More specifically, let $e^* = \{u^*, v^*\}$ and $f^* = \{x^*, y^*\}$. Edge f is included in G_e if and only if $d(u^*, x^*) + d(v^*, y^*) + 2 \geq k$ and $d(u^*, y^*) + d(v^*, x^*) + 2 \geq k$. A solution for the all-pairs shortest path problem of G^* will suffice for the distances required in computing G_e for all $e \in E(G)$.

Theorem 3.1.1 *[102] Given a planar graph G of n vertices and integer $k \geq 0$, ST Procedure decides if G has carvingwidth at least k or not using graph $H(G, k)$ in $O(n^2)$ time and $O(n^2)$ bytes of memory.*

3.2 Observations for improvements

We take the advantage of a set of observations on the game $RC(G, k)$ in [102] for an efficient implementation of ST Procedure. These improvements are based on deleting more *losing states* in the bipartite graph $H(G, k)$ representing the game $RC(G, k)$. A state is called losing state if rat will lose the game at that state. ST Procedure defines a state (r, v) a losing state if $v \in V(r)$, and deletes them in the initial state. The first improvement is to use the following lemma proved in [18] we are able to delete more losing states in the initial state of ST Procedure.

Lemma 3.2.1 *[18] For a face r and a vertex v in graph G with maximum node degree smaller than k, (r, v) is a losing state if there exist two faces s and t incident to v such that there are:*

1. *a closed walk W_1 in G^* with length smaller than k that consists of the shortest path from v_r^* to v_s^*, the clockwise walk from v_s^* to v_t^* around r_v^*, and the shortest path from v_t^* to v_r^* ; and*
2. *a closed walk W_2 in G^* with length smaller than k that consists of the shortest path from v_r^* to v_s^*, the counter-clockwise walk from v_s^* to v_t^* around r_v^*, and the shortest path from v_t^* to v_r^*.*

Once the shortest paths from v_r^* to all other vertices of G^* have been computed, checking if (r, v) is a losing state by the conditions of Lemma 3.2.1 is proportional to the node degree of v. Therefore, it takes $O(n)$ time to verify (r, v) for a face r and all $v \in V(G)$. For each face r, let $U(r)$ be the set of vertices that for every $v \in U(r)$, (r, v) is a losing state computed by the sufficient condition of Lemma 3.2.1. From Theorem 3.1.1, we have the following result.

Theorem 3.2.1 *[18] Given a planar graph G of n vertices and $k \geq 0$, ST Procedure decides if G has carvingwidth at least k in $O(n^2)$ time and $O(n^2)$ bytes of memory when the losing states $(r, v), v \in U(r)$, are deleted at the initial step of the deletion process for each face r.*

The next improvement is achieved by decreasing the size of $H(G, k)$ through the addition of new states that each replaces a set of states in $H(G, k)$. For each face $r \in R(G)$, we define G_r to be the subgraph of G obtained by deleting the noisy edges from G when the rat-catcher is in face r. Recall that G_e is the quiet subgraph of G when the rat-catcher is on edge e. One observation is that every quiet edge in G_r will stay quite in G_e and therefore, $E(G_r) \subseteq E(G_e)$. Based on this, a component vertex v_r of G_r is in a subgraph of a component of G_e. Hence, when the rat-catcher moves from face r to edge e and the rat is at any vertex of a component D of G_r, the component of G_e on which the rat can move is the same one that contains D as a subgraph. Thus, instead of using (r, v) in $H(G, k)$, we can work on the connected components of G_r. Therefore, the set of states $(r, v_1), (r, v_2), ..., (r, v_m)$ that are in the same component D of G_r can be replaced by (r, D). The game $RC(G, k)$ can be described by a bipartite graph $H'(G, k)$, where the vertex set of $H'(G, k)$ is $S' \cup T$ and S' is defined as

$$S'_r = \{(r, D) | D \text{ is a connected component of } G_r\}. \tag{3.1}$$

There is an edge between $(r, D) \in S'$ and $(e, C) \in T$ if face r is incident to edge e and D is a subgraph of C. In the new bipartite graph $H'(G, k)$, a state (r, D) is a losing state if for every vertex $v \in D$, (r, v) is a losing state. Similarly, if (e, C) is a losing state then for every face r incident to e and every component D of G_r that is a subgraph of C, (r, D) is a losing state. Summarizing the above observations and from Lemma 3.2.1, the following theorem holds.

Theorem 3.2.2 *[18] Given a planar graph G of n vertices and $k \geq 0$, ST Procedure decides if G has carvingwidth at least k using graph $H'(G, k)$ in $O(n^2)$ time and $O(n^2)$ bytes of memory.*

The third improvement can be achieved by reducing calculations in ST Procedure whose results may not be used in the following steps. In ST Procedure for every edge e, X_e is computed in the beginning of the procedure. However, the elements of X_e for an edge e incident to faces r and r' at a step of ST Procedure can be computed in $O(n)$ time from the

elements of X_r and $X_{r'}$ at that step. This gives an option for implementing ST Procedure that does not store X_e but dynamically computes it from X_r and $X_{r'}$ during the deletion process. The down side of computing X_e dynamically is that if for some face r, X_r is updated then for every incident edge e to r, T_e must be computed. This observation can be expressed as the following theorem proved in [18].

Theorem 3.2.3 *[18] Given a planar graph G of n vertices and $k \geq 0$, ST Procedure can decide if G has carvingwidth at least k or not in $O(n^3)$ time and $O(n^2)$ bytes of memory if for each edge e, X_e is not kept but dynamically computed during the deletion process.*

Finally, it is easy to see that if all states of S_r (or S'_r) for some face r are losing states then the rat-catcher wins the game.

Observation 3.2.1 *[18] If X_r becomes empty for some face r during the deletion process then graph G has carvingwidth smaller than k.*

By this observation we can terminate ST Procedure when some X_r becomes empty. This may save the computation time when the rat-catcher wins the game.

3.3 Efficient implementations

Let G be a connected planar graph with a given embedding and $V(G) = \{v_1, ..., v_n\}$. We first describe a straightforward implementation (called Naive) of ST Procedure and then propose several improvements on the Naive implementation. Those improvements try to reduce both the memory space and the running time of ST Procedure.

3.3.1 Naive implementation

A straightforward implementation of ST Procedure would use graph $H(G, k)$ for deciding the outcome of the game $RC(G, k)$. We use the following data structure for graph $H(G, k)$ in Naive.

- For each face $r \in R(G)$, A Boolean array B_r (of n elements), and is assigned such that $B_r[i]$ is used to indicate if $(r, v_i) \in X_r$ or not. A list of $|E(r)|$ elements is used to keep the edges incident to r.

- For each edge $e \in E(G)$, the two faces r and r' incident to e are kept. All components of G_e are kept in a list. Each component of G_e is given an index and component C_j is kept in the jth element of the list. The element of the list for C_j contains the set of vertices of C_j, $c(r, e, C_j)$, $c(r', e, C_j)$, and a Boolean variable indicating if (e, C_j) has been deleted from X_e or not. An integer array I_e (of n elements) is used to indicate which component a vertex is in. If v_i is a vertex of C_j then $I_e[i]$ is set to j.

- In addition to the face and edge data, a stack L is used and a distance matrix is kept for the all pairs shortest distances in the dual graph G^* of G.

Based on the above data structures, it is easy to verify that the Naive implementation runs in $O(n^2)$ time [18]. A simple calculation shows that Naive implementation requires about $40n^2$ bytes of memory when G is a medial graph. Since there are many single vertex components in G_e for an operating system with a minimum memory allocation of size 16 bytes, the memory usage in practice is close to $50n^2$ bytes. For the instances tested, Naive implementation requires a slightly smaller memory size than that of the straightforward implementation in [68]. It can solve instances with size up to 5,000 edges using about 1 GB of memory. The memory space required by the straightforward implementations could become a bottleneck for solving large instances.

3.3.2 Common improvements

In this section we describe two common improvements that are used in all of our efficient implementations of ST Procedure.

The first improvement is based on restricting the rat-catcher to move within the faces of a subset $Q \subset R(G)$. We start with a small Q (including one face) and perform the deletion process for the bipartite graph corresponding to the subgraph induced by Q until no deletion is possible. Then we enlarge Q by adding a new face to Q and repeat the deletion process. To add a new face to Q, a face whose losing states has been updated is given a higher priority to put to the stack. Q is enlarged until $Q = R(G)$. According to Observation 3.2.1, ST Procedure may stops at a small Q when the rat-catcher wins the game. Also, for a given subset Q, the losing states are deleted from X_r and $X_e (r \in Q, e \in E(r))$, then the data for X_r and X_e can be compressed before Q is enlarged. This helps in reducing the time and memory of ST Procedure.

In the second common improvement we reduce the memory usage through parsimonious data structure for edge data. We try to keep the minimum data for each edge. More precisely, instead of keeping a list of all components, we keep only a list of components with at least one edge. $I_e[i]$ is used to indicate if the component containing single node i has been deleted from X_e or not. This helps in reducing time and memory of ST Procedure. We also observe that if there are constant number (c) of small components (with constant fraction of n vertices (σn)), we do not need to keep the sets of vertices of the components in the list. We can use I_e to find the vertices of the component. Since c and σ are constants, this improvement does not increase the time complexity. Clearly a smaller σ saves more memory but may results in a larger running time. Similarly, a larger c saves more memory but may increase the running time. We have chosen $\sigma = 1/100$ and $c = 100$ in this study. A distance matrix is used to store the all-pairs shortest distances. We decide the integer type for the distance matrix based on the input integer k to ST Procedure. When G is a medial graph, we can reduce the required memory size to about $4n^2$ bytes if one-byte integer arrays are used for each I_e and the distance matrix. The required memory is $7n^2$ bytes when two-bytes integer arrays are used.

3.3.3 More improvements

Improvement A_1

This improvement is based on Theorem 3.2.1. In Improvement A_1, the elements $(r, v), v \in U(r)$, are deleted from X_r and put to the stack at the initial step of the deletion process for face r. Since $|U(r)|$ is usually much larger than $|V(r)|$, A_1 gives a room for improving both the running time and memory space.

Improvement D_1

The features of D_1 can be expressed by dynamic data creation and data compression. In D_1 the data for a face (edge) are created only when ST Procedure starts to process the face (edge). When some losing states are deleted, the face/edge data are compressed. In the naive implementation of ST Procedure, the distance matrix of the graph is computed in the first step, and to compute the distance matrix for every vertex v_r^* we need to solve $|R(G)|$ single source shortest path problems. In D_1 the distance matrix is discarded. When we process a face r, we create the data for r and the data for I_e for all e incident to r. Since each edge is incident to two faces r and r', the total number of single source shortest path calculations is bounded by $2|E(G)|$. When G has n vertices and is a medial graph,

$|R(G)| = n + 2$ and $|E(G)| = 2n$. From this, if the distance matrix is used, we need to solve $n + 2$ single source shortest path problems while we need to solve at most $4n$ single source path problems if D_1 is applied.

Combining D_1 with A_1, the required memory size is now about $5n \times q$ bytes if one-byte integer arrays are used for I_e and about $9n \times q$ if two-byte integer arrays are used, where q is the average of $|V(G) \setminus U(r)|$. For the Delaunay triangulation instances tested in this study, q is less than $0.3n$ (instances dependent). A_1 and D_1 are basic improvements. Our implementations based on the following improvements always use A_1 and D_1 as well.

Improvement A_2

Improvement A_2 is based on Theorem 3.2.2. For each face r, instead of S_r, A_2 initializes X_r to include all states of S'_r. For each face r, we need a Boolean array B_r of size $|S'_r|$ which is usually smaller than n. Similarly, for each edge e incident to faces r nad r', the sizes of the integer arrays I_e and I'_e are reduced accordingly. Combined with the basic improvements, A_2 can reduce the memory size significantly. A_2 can save memory space but may increase the running time a little since we need to compute the connected components for each face.

Improvement A_3

A_3 is based on Theorem 3.2.3 and performs re-calculation for edge data. A_3 keeps the face data once they are created but keeps the edge data for only a pre-defined maximum number of edges. Once this number is reached A_3 starts to delete the entire X_e for some edge e. If a deleted X_e is needed again, X_e is re-computed from X_r, where r is incident to e.

Improvement D_2

In D_2, we use a bit vector B_r for the data of face r, with one bit for one element of X_r. The memory size for face data is 1/8 of that when a one-byte Boolean array is used. But more complex bit operations have to be used. The extra running time due to the bit operations is negligible. Combined with A_3, this improvement is especially effective on memory saving because when A_3 is applied the memory used by the face data may become dominating. It is easy to check that all improvements except A_3 do not change the order of running time of ST Procedure. However, applying A_3, the running time of ST Procedure may become $O(n^3)$.

3.4 Computational study

All of our efficient implementations use common improvements. These implementations always use A_1 and D_1 improvements with any of A_2, A_3 and D_2 improvements. We do not mention A_1 and D_1 explicitly in those implementations. We test our implementations on the following three classes of instances.

- Class (1) of instances includes Delaunay triangulations of point sets taken from TSPLIB [95]. The instances are provided by Hicks and are used as test instances in the previous studies [68, 69].
- The instances in Class (2) are generated by the LEDA library [1, 88]. LEDA generates two types of planar graphs. One type of the graphs are the randomly generated maximal planar graphs and their subgraphs obtained from deleting a set of edges. Since the maximal planar graphs generated by LEDA always have branchwidth four (see Appendix A), the subgraphs obtained by deleting edges from the maximal graphs have branchwidth at most four. The graphs of this type are not interesting for the study of branchwidth and branch-decompositions. The other type of planar graphs are those generated based on a set of geometric properties, including Delaunay triangulations and triangulations of points uniformly distributed in a two-dimensional plane, and the intersection graphs of segments uniformly distributed in a two-dimensional plane. We will report the results on the intersection graphs.
- The instances in Class (3) are generated by the PIGALE library [2]. PIGALE randomly generates one of all possible planar graphs with a given number of edges based on the algorithms in [101].

TSPLIB is a library of sample instances for the Travel Salesman Problem (TSP) and related problems from various sources and of various types. This library provides a collection of benchmark instances of varying difficulties, which has been used by many research groups for comparing computational results. Random planar grphas generated by LEDA includes some geometric properties that are interested in many Geometric problems. This class of instances is used in many computational studies such as [4, 6, 15, 39]. LEDA does not generate random planar graphs based on the uniform distribution however, PIGALE generates random planar graphs with uniform distribution which is particularly intended for graph theoretical research problems.

We use our implementations to compute the carvingwidth of the medial graphs of the input instances (i.e., the input graph to ST Procedure is not an instance itself but the medial graph of the instance). Our implementations are tested on a computer with Intel(R) Xeon(TM) 3.06GHz CPU, 2 GB physical memory and 8 GB swap memory. The operating system is SUSE LINUX 10.0, and the programming language used is C++.

To compute the branchwidth of G an initial guess is needed for k. It is known that the branchwidth of a planar graph of n vertices is at most $\sqrt{4.5n}$ [56]. From this, $2\sqrt{4.5n}$ is an upper bound on the carvingwidth of the medial graph of an instance of n vertices. Following a similar approach in [68], another upperbound l is defined as the twice as the minimum eccentricity among all vertices in the dual of medial graph. Finally, we take $min\{2\sqrt{4.5n}, l\}$ as the initial value for k.

Either the linear search or the binary search can be used to find the carvingwidth starting from the initial guessed k. In the linear search, when the rat-catcher wins, k is decreased by two and ST Procedure is re-called until the rat wins the game. In the binary search, the ST Procedure is called to search for the carvingwidth between k (upper bound) and the node degree of $M(G)$ (which is four and a lower bound). For the instances in classes (1) and (2), the eccentricity-based guess is very close to the carvingwidth and k always takes the value of l. The linear search uses a smaller number of iterations to find the carvingwidth than the binary search. For instances in Class (3), the eccentricity-based guess could be very large and k may take $2\sqrt{4.5n}$ for large instances. Since $2\sqrt{4.5n}$ is still far away from the carvingwidth, the binary search does a better job in this case. One may run the linear search and binary search in parallel and take the results from the one which finishes earlier. In the following sections, we report the computational results in a number of tables. In the tables, for each instance, the number of edges is the number of vertices of the medial graph of the instance (the input graph to ST Procedure), bw is the branchwidth of the instance that is the half of the carvingwidth of the medial graph of the instance, and Itr is the number of iterations (calls of ST Procedure) to find the carvingwidth. An X in the tables indicates that the implementation runs out of physical memory (requires more than 2 GB of memory) for solving that instance.

3.4.1 Results for instances in Class(1)

Table 3.1 and Table 3.2 show the computation time and memory usage of Naive and efficient implementations for the carvingwidth of medial graphs of instances in Class(1). One-byte

integer arrays are used for each edge and the distance matrix. The most time efficient implementation is A_1 which is faster than Naive by a factor of at least 10 and uses at most 1/10 memory of Naive. Data compression of D_1 can reduce the memory usage of A_1 by factor of $1/3 \sim 1/4$. Recall that A_2 and D_2 are effective in reducing the memory size and the results show that A_2D_2 is the most memory efficient implementation with time complexity $O(n^2)$. Improvement A_3 has time complexity $O(n^3)$ and its performance depends on the maximum number of edges that are kept. In our implementation we keep at most 500 edges in A_3. Among all implementation $A_2A_3D_2$ is the most memory efficient one, and it is faster than Naive by factor of $6 \sim 7$. Although implementation $A_2A_3D_2$ has time complexity $O(n^3)$, its running time is at most as twice as the implementation A_2D_2 for this class of instances.

3.4.2 Comparison with previous works

As mentioned earlier, previous studies show that memory usage of ST Procedure is a major limiting factor. In [68] Hicks proposes two memory friendly implementations of ST Procedure which are named *comprat* and *memrat* in Table 3.3. The results shown in Table 3.3 are the results of these implementations using instances of Class(1) on a SGI Power Challenge with 6×194 MHz processors, 1 GB of physical memory and 1Gbyte of swap memory. Table 3.3 includes Hick's results and ours. The straightforward implementation of ST Procedure is called *rat* in this table. As discussed earlier Hick's experimental results and ours are performed on separate computational platforms due to the lack of Hick's implemented software. The platforms used to produce Hick's results has 194MHz CPU's [68] while the platforms used to compute our results has 3.06GHz CPUS. However, this difference is largely irrelevant when comparing the memory usage between the two methods. The results show that for large instances our implementations are faster by factor of at least 100 than memtrat and comprat. The memory usage of the largest instance reported in [68] (brd14051) is 600 MB. For the same instance A_2D_2 uses about 1/4 and $A_2A_3D_2$ uses about 1/8 of the memory than the memrat memory usage.

3.4.3 Results for instances in Classes (2) and (3)

Similar to the results for Class (1) instances, implementation A_1 is the most time efficient one among all implementations. In addition, A_2D_2 is the most memory efficient implementation

graph G	$\|E(G)\|$	bw	Itr	computation time (in seconds)								
				Naive	A_1	A_1D_1	A_2	A_2D_2	A_3	A_3D_2	A_2A_3	$A_2A_3D_2$
pr1002	2972	21	2	51.2	4.23	5.38	6.07	6.35	5.58	5.83	6.52	6.98
rl1323	3950	22	2	95.7	6.47	8.0	9.19	9.56	8.65	9.05	12.3	13.1
d1655	4890	29	2	158	11.3	15.0	17.3	17.6	15.7	16.7	21.6	22.3
rl1889	5631	22	2	195	13.8	16.6	20.1	20.8	19.2	20.4	29.5	30.5
u2152	6312	31	3	X	24.5	25.8	32.1	32.8	31.8	31.3	49.1	50.4
pr2392	7125	29	2	X	21.1	25.8	32.1	32.8	31.8	31.3	49.1	50.4
pcb3038	9101	40	2	X	31.6	41.3	50.8	51.9	44.6	49.7	83.2	74.2
fl3795	11326	25	2	X	63.7	80.2	99	104	86.3	98	155	163
fnl4461	13359	48	2	X	67.4	92.4	116	119	97.1	110	185	185
rl5934	17770	41	2	X	151	197	245	249	213	241	385	391
pla7397	21865	33	2	X	246	296	376	385	393	453	606	629
usa13509	40503	63	4	X	X	1061	1359	1371	1241	1386	2154	2165
brd14051	42128	68	2	X	X	1061	1418	1417	1226	1361	2274	2282
d15112	45310	78	3	X	X	2070	2810	2852	2379	2598	4549	4603
d18512	55510	88	2	X	X	X	2315	2321	2100	2241	3752	3756
pla33810	101362	100	5	X	X	X	12379	12614	X	14747	20482	21734

Table 3.1: Computation time (in seconds) of Naive and efficient implementations for Class (1) instances.

graph G	$\|E(G)\|$	Maximum Memory usage (Mbyte)								
		Naive	A_1	A_1D_1	A_2	A_2D_2	A_3	A_3D_2	A_2A_3	$A_2A_3D_2$
pr1002	2972	413	39	16	8	8	10	9	8	7
rl1323	3950	734	66	23	11	10	15	13	11	9
d1655	4890	1188	99	30	14	11	17	14	13	10
rl1889	5631	1424	130	46	18	14	28	21	16	12
u2152	6312	X	161	41	17	14	26	20	16	13
pr2392	7125	X	204	66	21	17	35	26	19	15
pcb3038	9101	X	328	76	25	21	36	27	22	19
fl3795	11326	X	504	132	58	42	66	63	40	23
fnl4461	13359	X	698	158	39	32	66	43	33	26
rl5934	17770	X	1226	358	67	51	155	86	50	35
pla7397	21865	X	1850	436	123	85	238	144	83	44
usa13509	40503	X	X	1534	220	153	498	271	149	79
brd14051	42128	X	X	1600	215	149	580	283	149	82
d15112	45310	X	X	1795	227	156	508	256	156	86
d18512	55510	X	X	X	284	198	706	328	194	106
pla33810	101362	X	X	X	814	508	X	876	507	198

Table 3.2: Memory usage (in megabytes) of Naive and efficient implementations for Class (1) instances.

graph G	$\|E(G)\|$	bw	Itr	computation time (in seconds)		
				rat	comprat	memrat
pr1002	2972	21	2	338	448	562
rl1323	3950	22	3	876	1519	1590
d1655	4890	29	3	1318	1608	2206
rl1889	5631	22	3	M	3931	4012
u2152	6312	31	4	M	3207	4704
pr2392	7125	29	3	M	3813	5167
pcb3038	9101	40	4	M	13817	15865
fl3795	11326	25	3	M	18469	17142
fnl4461	13359	48	4	M	35933	51305
rl5934	17770	41	3	M	73468	66461
pla7397	21865	33	2	M	65197	53564
usa13509	40503	63	1/2	M	M	413861
brd14051	42128	68	3	M	M	594684

Table 3.3: Computation time (in seconds) of rat, comprat, and memrat quoted from Table 1 of [68]

with time complexity $O(n^2)$ while, $A_2A_3D_2$ is the most memory efficient one among all implementation for the instances in these two classes. Table 3.4 shows the computation time and memory of Naive and Implementations A_1, A_2D_2, and $A_2A_3D_3$ for instances of intersection graphs of segments generated by LEDA.

We choose a function from PIGALE library, for generating 2-connected planar instances of Class(3). Given the number m edges the function randomly generates one of all possible 2-connected planar graphs of m edges. Since the generated graphs have parallel edges and they are not interesting in ST Procedure, we remove the resulted graph has m' edges. For this class of instances, the eccentaricity-based initial guess is usually bad and $2\sqrt{4.5n}$ is used. Since the initial guess is the upper bound and much larger than the actual carvingwidth, binary search always finishes earlier than linear search. Although, we used two-byte integer arrays for edge data for this class, the memory usage for the PIGALE instances is very small comparing to the instances of Class(1) and Class(2). Table 3.5 gives the computation time and memory of Naive and Implementations A1 , A2 D2 , and A2 A3 D2 .

graph G	$\|E(G)\|$	bw	Itr	Time (in seconds)				Memory(MByte)			
				Naive	A_1	A_2D_2	$A_2A_3D_2$	$Naive$	A_1	A_2D_2	$A_2A_3D_2$
rand1300	2030	7	6	51.1	10.4	13.1	16.3	181	32	8	8
rand1900	3029	8	5	102	15.7	18.7	25.8	389	68	13	12
rand3050	5032	9	4	283	32.5	34.2	61.5	1179	180	32	22
rand6000	10261	12	2	X	95.5	101	147	X	724	92	41
rand8700	15090	14	3	X	292	370	849	X	1559	160	71
rand1500	20279	13	2	X	X	557	2002	X	X	269	120
rand1700	30433	14	2	X	X	1334	3190	X	X	529	216
rand22500	40622	18	2	X	X	2156	2760	X	X	758	326
rand28000	50901	18	3	X	X	4002	6993	X	X	1112	494
rand33000	60398	20	2	X	X	5633	8540	X	X	1472	624
rand54000	100037	22	2	X	X	X	21417	X	X	X	1382

Table 3.4: Computation time and memory of intersection graphs of segments generated by LEDA.

m	graph G	$\|E(G)\|$	bw	Itr	Time (in seconds)				Memory(MByte)			
					Naive	A_1	A_2D_2	$A_2A_3D_2$	$Naive$	A_1	A_2D_2	$A_2A_3D_2$
2400	PI1180	2022	7	5	23	6.73	9.08	10.4	196	32	8	7
	PI1182	2016	7	5	22.6	7.44	10.4	12.1	190	32	8	7
	PI1186	2029	6	5	23.5	6.56	9.24	10	205	32	7	7
	PI1193	2019	6	5	19.1	7.39	10.5	11.1	156	32	7	7
	PI1207	2029	9	5	30.6	5.41	6.64	7.04	167	32	7	7
6000	PI2995	5043	7	6	156	58.7	93.3	96.5	1034	178	14	13
	PI2996	5015	8	7	210	72.1	115	117	1119	176	16	14
	PI2998	5049	7	6	163	58.3	92.4	102	1090	178	17	15
	PI3017	5063	8	7	197	70.9	111	117	1112	179	15	14
	PI3018	5074	7	6	158	57.4	88.2	95.5	986	180	15	14
12000	PI5940	10016	7	8	X	289	522	563	X	683	28	26
	PI5992	10101	7	8	X	286	558	580	X	695	27	25
	PI5998	10144	7	8	X	304	555	583	X	701	26	24
	PI6043	10146	8	8	X	299	550	576	X	701	30	27
	PI6067	10173	7	8	X	312	555	584	X	705	26	25
18000	PI8950	15097	10	9	X	907	1771	2089	X	1541	48	39
	PI8977	15065	9	9	X	913	1791	1971	X	1535	43	38
	PI8986	15058	8	8	X	765	1559	1643	X	1533	42	39
	PI8995	15080	9	9	X	885	1787	1911	X	1538	41	38
	PI9020	15053	8	8	X	807	1582	1688	X	1532	41	37
24000	PI11974	20071	9	9	X	X	3646	3702	X	X	46	44
35000	PI17495	30003	7	9	X	X	8597	8610	X	X	70	67
46000	PI22640	40074	5	9	X	X	14163	14210	X	X	94	89
56000	PI27671	50095	8	10	X	X	24684	24702	X	X	118	114
66000	PI32943	60634	8	10	X	X	36909	37076	X	X	156	138
76000	PI37730	70022	6	10	X	X	49136	49180	X	X	188	160

Table 3.5: Computation time and memory of random instances generated by PIGALE.

graph	$\|E(G)\|$	Computation time (seconds) for k							
		$k = 2(bw+1)$				$k = 2bw$			
G		Naive	A_1	A_2D_2	$A_2A_3D_2$	$Naive$	A_1	A_2D_2	$A_2A_3D_2$
rl1889	5631	87.6	0.196	0.526	0.545	79.2	8.55	17.5	27.2
usa13509	40503	X	X	41.9	61.2	X	X	1118	1888
d15112	45310	X	X	687	1153	X	X	1578	2632
rand3050	5032	59.4	4.33	5.43	10.7	48.4	20.7	22.7	42.7
rand22500	40622	X	X	9.2	14.4	X	X	1714	2368
rand33000	60398	X	X	1203	1789	X	X	3825	6300
PI2995	5043	12.1	10.2	17.4	18	10.3	9.91	17.5	18.3
PI22640	40074	X	X	1670	1680	X	X	1673	1676
PI32943	60634	X	X	3725	3727	X	X	3471	3471

Table 3.6: Computation time (in seconds) of several implementations for k close to the carvingwidth 2bw.

3.4.4 Computation time of one iteration

ST Procedure is usually called multiple times to find the carvingwidth of a planar graph. The number of calls (iterations) is instance dependent. In computing the branch-decompositions, the computation time of one iteration is an important measure for the time efficiency. Table 3.6 shows the computation time of Naive and Implementations A_1, A_2D_2, and $A_2A_3D_2$ in the iteration when the rat wins the game and the iteration when the rat-catcher wins the game with the noisy level k closest to the carvingwidth for some instances in Classes (1), (2), and (3). From Observation 3.2.1, the deletion process of ST Procedure may terminate earlier when the rat-catcher wins the game. It can be seen from the table that ST Procedure generally uses much less time when the rat-catcher wins for instances of Classes (1) and (2). For instances in Class (3), the computation time is not much different between the cases where the rat wins from those where the rat-catcher wins. The reason is that the time of the deletion process is not a dominating part of the total running time.

3.5 Computing an optimal branch-decomposition

ST Procedure decides if the branchwidth of a planar graph is at most k, but this algorithm does not find the corresponding branch-decomposition if it exists. Using ST Procedure as a subroutine, Seymour and Thomas in [102] proposed an algorithm for constructing a minimum-width branch-decomposition of a planar graph. This algorithm calls ST Procedure $O(n^2)$ times, that give rise to an $O(n^4)$ algorithm. Gu and Tamaki reduce the number of times that ST Procedure is called to $O(n)$ and therefore, resulting in an $O(n^3)$ algorithm.

3.5.1 Seymour and Thomas algorithm: Edge Contraction

Seymour and Thomas proposed an algorithm, known as *Edge Contraction (EC) method*, for computing an optimal branch-decomposition of a planar graph [102]. The contraction of an edge e in a graph G is to remove e from G, identify the two end vertices of e by a new vertex, and make all edges incident to e incident to the new vertex. Figure 3.1 gives an example of edge contraction.

We denote by G/e the graph obtained by contracting e in G. Given a 2-connected planar graph G, an edge e is *contractible* if G/e is 2-connected and the carvingwidth of G/e is at most the carvingwidth of G. The EC algorithm computes an optimal branch-decomposition

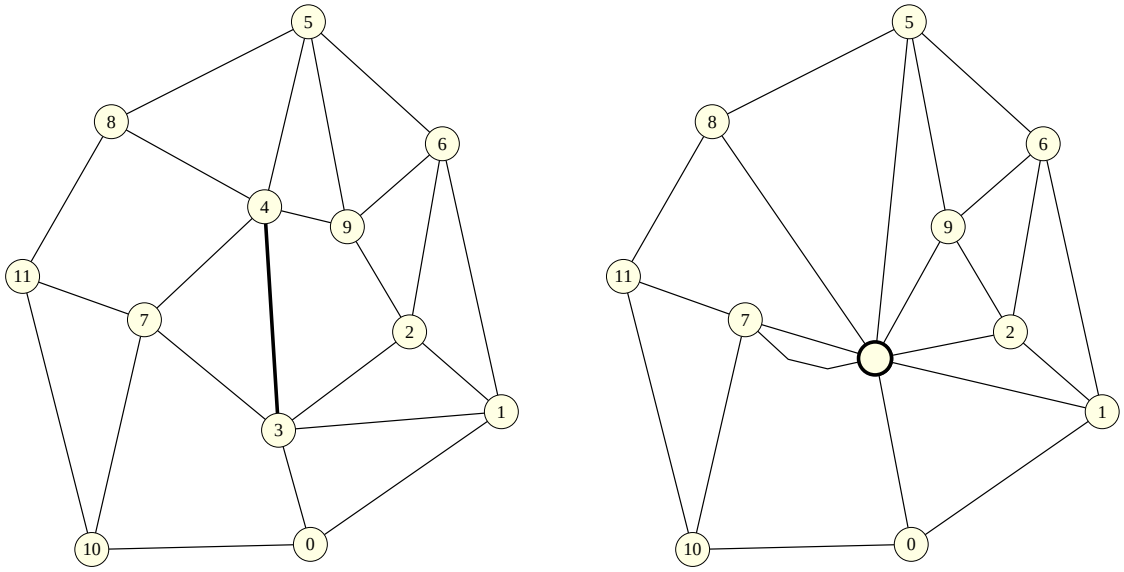


Figure 3.1: A planar graph with 12 nodes and resulting graph after contracting edge between vertices 3 and 4.

of G by a sequence of edge contractions of the medial graph $M(G)$ of G as follows:

1. The carvingwidth cw of $M(G)$ is computed by ST Procedure.
2. A contractible edge e of $M(G)$ is found by ST Procedure and $M(G)$ is contracted to graph $M(G)/e$.
3. Steps 2 and 3 are repeated on $M(G)/e$ until the graph has three vertices.
4. A carving decomposition of $M(G)$ with width at most cw is constructed based on the sequence of edge contractions.
5. The branch-decomposition of G is obtained from the carving-decomposition of $M(G)$.

It is proved in [102] that for any 2-connected planar graph there is a contractible edge. Since for a 2-connected planar graph G, $M(G)$ is 2-connected, the above algorithm computes the minimum branch-decomposition of every 2-connected planar graph. To verify whether an edge is contractible or not, ST Procedure is used to test if $M(G)/e$ has carvingwidth at most cw. In the worst case, all edges may be checked to find a contractible one. Thus, for a

graph of n vertices, the EC algorithm may call ST Procedure $O(n)$ times for one contraction and $O(n^2)$ times in total. So the time complexity of the algorithm is $O(n^4)$. In contracting edges, if an edge is identified as a contractible edge, the contractability test is called *positive call*, otherwise it is called a *negative one*.

3.5.2 Gu and Tamaki algorithm

Gu and Tamaki proposed an algorithm, called *GT Algorithm*, that uses a better strategy to find contractible edges in Step (2) of EC algorithm [61]. Once ST Procedure shows that an edge is not contractible then the edge will not be chosen to contract again unless a necessary condition for that edge to become contractible is satisfied.

Let $M(G)$ be an input planar graph. We define $\mathcal{M}^0$ be the set of all singleton of $V(M(G))$. Each singleton subset $X = \{x\}$ represents vertex x of $M(G)$. When an edge $e = \{x, y\}$ is contracted in $M(G)$, a new subset $X \cup Y$ is added into $\mathcal{M}^0$ to get $\mathcal{M}^1$. The new subset $X \cup Y$ in $\mathcal{M}^1$ represents the new vertex in $M(G) \setminus e$ identifying the two end vertices x and y of the contracted edge e. In general, for $i > 0$, let $e = \{x, y\}$ be an edge in the graph obtained from $i-1$ edge contractions in $M(G)$. When we contract e we compute $\mathcal{M}^i = \mathcal{M}^{i-1} \cup \{X \cup Y\}$. Note that X and Y are maximal subsets in $\mathcal{M}^{i-1}$. We call each $\mathcal{M}^i$ a *binary merging*.

Let $\mathcal{M}$ be a binary merging. A sequence $X_0 \supset X_1 \supset ... \supset X_m$ of elements of $\mathcal{M}$ is called a *chain* if each X_i for $1 \leq i \leq m$ is maximal proper subset of X_{i-1}. If we denote the above chain by C, then X_0 is called the top of C, denoted by $\top_C$, and X_m is called the bottom of C, denoted by $\bot_C$. Chain C is called a *barrier* if $|\delta_{M(G)}(X_0 \setminus X_m)| > k$. The necessary condition to check the contractability of an edge is based on the following lemma:

Lemma 3.5.1 (barrier lemma)*[61] Let $\mathcal{M}$ be a binary merging in V with width k or smaller and X, Y two maximal elements of $\mathcal{M}$. Suppose there is a binary merging $\mathcal{M}' \subseteq \mathcal{M}$ and maximal elements X' and Y' of $\mathcal{M}'$ with $X' \subseteq X$ and $Y' \subseteq Y$ such that the following conditions hold:*

1. *X and Y are contractible in $\mathcal{M}$;*
2. *$|\delta(X' \cup Y')| \leq k$;*
3. *$E_G(X \setminus X', Y \setminus Y') = \emptyset$;*

4. *there is no barrier* B *in* $\mathcal{M}$ *with* $\top_B = X$ *and* $\perp_B \supseteq X'$;

5. *there is no barrier* B *in* $\mathcal{M}$ *with* $\top_B = Y$ *and* $\perp_B \supseteq Y'$;

then, X' *and* Y' *are contactable in* $\mathcal{M}'$.

Using the above lemma, GT Algorithm avoids the repeated negative tests on a same edge, and thus calls the ST Procedure only $O(n)$ times reducing the time complexity to $O(n^3)$ for computing an optimal branch-decomposition of a planar graph. Gu and Tamaki suggest to use a new parsimonious contractability test to verify if two maximal elements are contractible in EC algorithm. In more details, we assume that all unordered pairs of elements on which ST Procedure is performed and the result was negative are stored in a list Q. To decide an edge $e = \{x, y\}$ corresponding to two elements X and Y in $\mathcal{M}$ is contractible, GT Algorithm works as follows:

- If $|\delta(X \cup Y)| > k$ then report "The edge is not contractible".
- Otherwise, if there is not any X' and Y' of $\mathcal{M}$ with $X' \subseteq X$ and $Y' \subseteq Y$ such that Q contains one of pairs: $\{X', Y\}, \{X', Y'\}, \{X, Y'\}$, then call ST Procedure.
- Otherwise, if conditions 2 to 5 of the above lemma hold then report "The edge is not contractible", otherwise call ST Procedure

It is proved that the last step based on the above lemma, saves $O(n)$ times of calling ST Procedure [61].

3.6 Computing optimal branch-decomposition: Efficient Implementation

In this section we describe our improvements for efficient implementation of algorithms to compute an optimal branch-decomposition. These improvements can be applied in both EC Algorithm and GT Algorithm. The first improvement is achieved by finding a better method for choosing an edge to contract. The second improvement is achieved by simplifying the input graph and reduce the running time.

Choosing an edge to contract Recall from the beginning of this chapter that no specific method has been identified to choose an edge to contract in EC or GT algorithms.

We study the performance of Round robin, Random and Greedy methods to choose a contracting edge:

- **Round robin:** We choose an edge to contract in a cyclic order in the list of edges.
- **Random:** We choose a random edge to contract.
- **Greedy:** We choose an edge $e = \{u, v\}$ with minimum $|\delta(u, v)|$ to contract.

Simplifying the graph We can reduce the size of the graph G without changing the optimal branch-decomposition. We define *simple path* P as a sequence of vertices $v_1, v_2, ..., v_l$ such that there is an edge between every v_i and v_{i+1} for $1 \leq i < l$ and the degree of every node v_i, $2 \leq i < l$, is two. Let $e_1, e_2, ..., e_{l-1}$ be the set of edges in P. We define $S(P)$ as the set of vertices common between P and $G \setminus P$. It is clear that $|S(P)| = 2$. Replacing a simple path by an edge in computing branch-decomposition does not affect the branchwidth and the optimal branch-decomposition.

Therefore, we search the medial graph and replace every simple path with an edge.

3.7 Computational study

We compared the EC Algorithm and GT Algorithm for instances from the same classes which are tested for ST Procedure. The testing platform has an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ (2.4 GHz) and 3 GB memory. The operating system is SUSE Linux 10.2 and the programming language used is C++.

3.7.1 Computational study: Edge contraction

We study the performance of EC algorithm on TSBLIB, Random and PIGALE graphs. Let $M(G)$ be the medial graph of the input graph G. $M(G)$ is represented by a list of edges l. The vertices of $M(G)$ need to be re-indexed after each contract operation. We used the methods described in the previous section for choosing an edge to contract. The following tables show the performances of these methods. In these tables, PosCall and NegCall are the number of positive calls and negative calls, respectively. Tables 3.7, 3.8 and 3.9 show the performance of Round Robin, Random and Greedy algorithms. The results show that Round Robin has the best performance among all methods. An X in the Tables 3.8 and 3.9 indicates that the program did not finish after 48 hours.

	Graph G	$\|E(G)\|$	bw	Round robin			
				Neg-Calls	Pos-Calls	Total Calls	time
(1)	pr1002	2972	21	105	2968	3073	3528.74
	rl1323	3950	22	118	3946	4064	8670.41
	d1655	4890	29	148	4886	5034	14577.2
	rl1889	5631	22	162	5627	5789	25450
	u2152	6312	31	339	6308	6647	29416.6
	pr2392	7125	29	260	7121	7381	52303.4
(2)	rand1408	1808	7	100	1804	1904	125.75
	rand1500	1994	6	96	1990	2086	617.83
	rand1300	1668	9	72	1664	1736	96.85
	rand1900	2580	8	60	2576	2636	1736.24
	rand3050	4456	10	66	4452	4518	18404.7
	rand6000	9332	12	477	9328	9805	177247
	rand8700	13924	13	748	13920	14668	195476
(3)	PI855	1434	6	75	1430	1505	694.23
	PI1277	2128	9	66	2124	2190	2154.21
	PI1467	2511	6	94	2507	2601	4118.44
	PI2009	3369	7	109	3365	3474	10143.2
	PI2518	4266	8	74	4262	4336	21818.5
	PI2968	5031	6	186	5027	5213	40524.7

Table 3.7: Number of calls for ST Procedure and computation time (in seconds) for Round-Robin method.

	Graph G	$\|E(G)\|$	bw	Random			
				Neg-Calls	Pos-Calls	Total Calls	time
(1)	pr1002	2972	21	817	2968	3785	21522.8
	rl1323	3950	22	6597	3946	10543	49442
	d1655	4890	29	1142	4886	6028	58717.9
	rl1889	5631	22	2276	5627	7903	80690.9
	u2152	6312	31	1642	6308	7950	84118.8
	pr2392	7125	29	3012	7121	10133	89485
(2)	rand1408	1808	7	16625	1804	18429	125.75
	rand1500	1994	6	23112	1990	25102	617.83
	rand1300	1668	9	21446	1664	23110	96.85
	rand1900	2580	8	11030	2576	13606	1691.16
	rand3050	4456	10	973	4452	5425	18404.7
	rand6000	9332	12	X	X	X	more than 42 hours
	rand8700	13924	13	X	X	X	more than 42 hours
(3)	PI855	1434	6	2304	1430	3734	11547.3
	PI1277	2128	9	5058	2124	7182	24230.7
	PI1467	2511	6	922	2507	3429	43256
	PI2009	3369	7	713	3365	4078	45815
	PI2518	4266	8	909	4262	5171	85363
	PI2968	5031	6	473	5027	5500	96671

Table 3.8: Number of calls for ST Procedure and computation time (in seconds) for Random method .

	Graph G	$\|E(G)\|$	bw	Greedy			
				Neg-Calls	Pos-Calls	Total Calls	time
(1)	pr1002	2972	21	5673	2968	8641	7635.48
	rl1323	3950	22	3061	3946	7007	9409.15
	d1655	4890	29	3272	4886	8158	127926
	rl1889	5631	22	7347	5627	12974	33936.8
	u2152	6312	31	9434	6308	15742	50068.9
	pr2392	7125	29	3870	7121	10991	78206
(2)	rand1408	1808	7	697	1804	2501	131.15
	rand1500	1994	6	1930	1990	3920	749.2
	rand1300	1668	9	921	1664	2585	117.44
	rand1900	2580	8	894	2576	3470	1736.24
	rand3050	4456	10	4203	4452	8655	28981.5
	rand6000	9332	12	X	X	X	more than 42 hours
	rand8700	13924	13	X	X	X	more than 42 hours
(3)	PI855	1434	6	141	1430	1571	824.1
	PI1277	2128	9	928	2124	3052	2410.91
	PI1467	2511	6	110	2507	2617	4690.46
	PI2009	3369	7	118	3365	3483	11829.5
	PI2518	4266	8	201	4262	4463	24192
	PI2968	5031	6	258	5027	5285	41725.74

Table 3.9: Number of calls for ST Procedure and computation time (in seconds) for Greedy method.

As mentioned in the previous section, removing simple paths in G does not change the branch-decomposition. We also study the performance of this graph simplification. Clearly, the performance is variable depending on the structure of the graph and the number of simple paths. We apply this simplification on the instances of all three classes. Table 3.10 shows the results for the instances of Class(3).

This simplification does not reduce the size of the graphs in Class(1) and Class(2) (results not shown). The computational results show that this graph modification can improve the running time by a factor of $2.92 \sim 16.16$ percent.

Computational study: Gu and Tamaki Algorithm

Gu and Tamaki proposed a set of data structures for their algorithm to achieve $O(n^3)$ time complexity. We use the same data structures in our implementation. In their algorithm a carving decomposition is defined as a binary merging and is generated gradually. The suggested data structures are two main data structures for representing the binary merging and the resulting graphs after contracting edges.

Let $\mathcal{M}^i$ and $M^i(G)$ be the binary merging and the resulting graph after contracting i

Graph	Round-Robin without simplification							Round-Robin with the path simplification						
G	Node	Edge	bw	Neg-Calls	Pos-Calls	Total Calls	time	Node	Edge	bw	Neg-Calls	Pos-Calls	Total Calls	time
PI855	855	1434	6	75	1430	1505	694.23	834	1422	6	68	1418	1486	649.16
PI1277	1277	2128	9	66	2124	2190	2154.21	1250	2101	9	80	2097	2177	1806.64
PI1467	1467	2511	6	94	2507	2601	4118.44	1455	2499	6	69	2495	2564	3897.68
PI2009	2009	3369	7	109	3365	3474	10143.2	1973	3333	7	119	3329	3448	9375.95
PI2518	2518	4266	8	74	4262	4336	21818.5	2482	4230	8	102	4226	4328	21181.95
PI2968	2968	5031	6	186	5027	5213	40524.7	2939	5002	6	189	4998	5187	37214.02

Table 3.10: Number of calls for ST Procedure when the path simplification is applied for the instances of Class (3).

edges, respectively. We maintain $\mathcal{M}^i$ and $M^i(G)$ for each $0 \leq i \leq n-1$. Every binary merging $\mathcal{M}^i$ constructed by the algorithm is represented by a binary forest. Every node of this forest is associated to a member X of $\mathcal{M}^i$, in the form of a sorted list, and the edge set $\delta_{M(G)}(X)$ also in the form of sorted list. Using this representation, contracting of two members of $\mathcal{M}^i$ can be done in $O(n)$ time.

$M^i(G)$ is defined as a graph whose nodes are the maximal elements of $\mathcal{M}^i$. There is an edge between nodes $X, Y \in V(M^i(G))$ if and only if $E_{M(G)}(X, Y) \neq \emptyset$. $M^i(G)$ is maintained in standard edge list representation. Every edge $e = \{X, Y\}$ of $M^i(G)$, $0 \leq i \leq n-1$ is associated with its corresponding edges in $M^{i-1}(G)$: if $X, Y \in V(M^{i-1}(G))$ this association is to the edges between X and Y in $M^{i-1}(G)$, otherwise if $X = X_1 \cup X_2$, with $X_1, X_2 \in V(M^{i-1}(G))$, say then this association is to the edges between X_1 and Y, if present, and to the edges between X_2 and Y, if present. The same condition is hold when $Y = Y_1 \cup Y_2$.

The graph instances and the computer that are used for this study are the same instances and the computer in the Edge contraction method.

Table 3.11 compares the EC method with GT algorithm. We compare the best results of EC, round robin method, with the results of GT Algorithm. The data in the table show that optimal branch-decompositions of planar graphs of a few thousands edges can be computed in a practical time. For most instances tested, repeated negative tests are not observed on any edge in the algorithm of edge contraction. So the advantage of GT Algorithm is not shown by those instances when the round robin edge selection is used. For some of the other edge selection heuristic, more repeated negative tests are observed in the algorithm of Seymour and Thomas. In such cases, GT Algorithm has much less negative calls and thus, runs faster than EC Algorithm.

	Graph G	$\|E(G)\|$	bw	Edge-Contraction Alg.				Gu-Tamaki Alg			
				Neg-Calls	Pos-Calls	Total Calls	time	Neg-Calls	Pos-Calls	Total Calls	time
(1)	pr1002	2972	21	105	2968	3073	3528.74	105	2968	3073	3621.7
	rl1323	3950	22	118	3946	4064	8670.41	118	3946	4064	8689.3
	d1655	4890	29	148	4886	5034	14577.2	146	4886	5032	14586.32
	rl1889	5631	22	162	5627	5789	25450	162	5627	5789	26147.4
	u2152	6312	31	339	6308	6647	29416.6	338	6308	6646	29384.1
	pr2392	7125	29	260	7121	7381	52303.4	260	7121	7381	53047.21
(2)	rand1408	1808	7	100	1804	1904	125.75	97	1804	1901	131.4
	rand1500	1994	6	96	1990	2086	617.83	96	1990	2086	646.1
	rand1300	1668	9	72	1664	1736	96.85	72	1664	1736	102
	rand1900	2580	8	60	2576	2636	1736.24	57	2576	2633	1694.8
	rand3050	4456	10	66	4452	4518	18404.7	64	4452	4516	18067.6
	rand6000	9332	12	477	9328	9805	177247	477	9328	9805	177412
	rand8700	13924	13	748	13920	14668	195476	747	13920	14667	196103
(3)	PI855	1434	6	75	1430	1505	694.23	75	1430	1505	704.5
	PI1277	2128	9	66	2124	2190	2154.21	57	2124	2181	1963.7
	PI1467	2511	6	94	2507	2601	4118.44	92	2507	2599	4026.9
	PI2009	3369	7	109	3365	3474	10143.2	102	3365	3467	10315
	PI2518	4266	8	74	4262	4336	21818.5	71	4262	4333	21809.3
	PI2968	5031	6	186	5027	5213	40524.7	186	5027	5213	40875.9

Table 3.11: Number of calls for ST Procedure and computation time (in seconds) for branch-decompositions.

Chapter 4

DOMINATING SET Problem in planar graphs

Recently, there has been significant theoretical progress towards fixed-parameter algorithms for the DOMINATING SET problem of planar graphs. It is known that the problem on a planar graph with n vertices and dominating number k can be solved in $O(2^{\sqrt{O(k)}}n)$ time using tree/branch-decomposition based algorithms. In this chapter, we report computational results of Fomin and Thilikos algorithm which uses the branch-decomposition based approach. The computational results show that the algorithm can solve the DOMINATING SET problem of large planar graphs in a practical time and memory space for the class of graphs with small branchwidth. For the class of graphs with large branchwidth, the size of instances that can be solved by the algorithm in practice is limited to about one thousand edges due to a memory space bottleneck. The practical performances of the algorithm coincide with the theoretical analysis of the algorithm. The results suggest that the branch-decomposition based algorithms can be practical for some applications on planar graphs.

4.1 Fomin and Thilikos algorithm

In this section we briefly review Fomin and Tilikos (FT) Algorithm. Readers may refer to [56] for more details. FT Algorithm solves the PLANAR DOMINATING SET problem of G in three steps. Step I computes a kernel H of G by the data reduction process such that

$size(H) \leq size(G), \gamma(H) \leq \gamma(G)$, and a minimum dominating set D of G can be computed from a minimum dominating set D' of H in linear time. Step II finds an optimal branch-decomposition T_B of H. Step III computes a minimum dominating set D' of H using the dynamic programming method based on T_B and constructs a minimum dominating set D of G from D'. In Step I, the data reduction rules introduced in [7] are used to decide if some vertices of G can be included in D or excluded for computing D. If a vertex v has been decided to be included in D, v is colored black. If v has been decided to be excluded for computing D in the future, v is removed from G. Every other vertex is colored grey. After the reduction process, we get a kernel $H(B \cup C, E)$, where B and C are the sets of black and grey vertices, respectively. The specific reduction rules will be introduced in the next section. To compute an optimal branch-decomposition T_B of H, either the edge-contraction algorithms [61, 102] or the divide-and-conquer algorithms [18] can be used. The divide-and-conquer algorithms are faster for large graphs in practice. In Step III, given a kernel $H(B \cup C, E)$, we find a minimum $D' \subseteq (B \cup C)$ such that $B \subseteq D'$ and D' dominates all vertices of C. As shown later, a minimum dominating set D of G can be constructed from D' in linear time. To compute D', first the branch-decomposition T_B of H is converted into a rooted binary tree by replacing a link $\{x, y\}$ of T_B by three links $\{x, z\}, \{z, y\}$, and $\{z, r\}$, where z and r are new nodes to T_B, r is the root, and $\{z, r\}$ is an internal link. For every internal link e of T_B, e has two child links incident to e. For every link e of T_B, let T_e be the subtree of T_B consisting of all descendant links of e. Let H_e be the subgraph of H induced by the edges at leaf nodes of T_e. To compute a minimum dominating set D' of H, we find all dominating sets (solutions) of H_e from which D' may be constructed for every link e of T_B by a dynamic programming method: the solutions of H_e for each leaf link e is computed by enumeration and the solutions for an internal link e is computed by merging the solutions for the child links of e. For a link of T, and separation $(A_e, \overline{A_e})$ induced by e, let $S_e = \partial(A_e)$. To find a solution of H_e, each vertex of S_e is colored by one of the following colors.

- **Black** denoted by 1, meaning that the vertex is included in the dominating set.
- **White** denoted by 0, meaning that the vertex is dominated at the current step of the algorithm and is not in the dominating set.
- **Grey** denoted by $\hat{0}$, meaning that we have not decided to color the vertex into black or white yet at the current step.

A solution of H_e subject to a coloring $\lambda \in \{0, 1, \hat{0}\}^{|S_e|}$ is a minimum set $D_e(\lambda)$ satisfying

- for $u \in B \cap S_e$, $\lambda(u)$ is black.
- every vertex of $V(H_e) \setminus S_e$ is dominated by a vertex of $\lambda(D_e)$; and
- for every vertex $u \in S_e$ if $\lambda(u)$ is black then $u \in D_e(\lambda)$, if $\lambda(u)$ is white then $u \notin D_e(\lambda)$ and u is dominated by a vertex of $D_e(\lambda)$.

Intuitively, $D_e(\lambda)$ is a minimum set to dominate the vertices of H_e with grey vertices removed, subject to the condition that the vertices of S_e are colored by λ. For every coloring $\lambda \in \{0, \hat{0}, 1\}^{|S_e|}$, $a_e(\lambda)$ is defined as $|D_e(\lambda)|$ if there is a solution of H_e subject to λ, otherwise as $+\infty$. For a leaf link e, colorings λ and sets $D_e(\lambda)$ are computed by enumeration. Assume that an internal link e has child links e_1 and e_2 in T_B. The colorings λ of S_e and sets $D_e(\lambda)$ are computed from the colorings λ_1 of S_{e_1}, sets $D_{e_1}(\lambda_1)$, colorings λ_2 of S_{e_2}, and sets $D_{e_2}(\lambda_2)$. Let $X_1 = S_e \setminus S_{e_2}$, $X_2 = S_e \setminus S_{e_1}$, $X_3 = S_e \cap S_{e_1} \cap S_{e_3}$, and $X_4 = (S_{e_1} \cup S_{e_2}) \setminus S_e$. A coloring λ of S_e is formed from λ_1 and λ_2 if:

1. For $u \in X_1$, $\lambda(u) = \lambda_1(u)$.
2. For $u \in X_2$, $\lambda(u) = \lambda_2(u)$.
3. For $u \in X_3$, if $\lambda_1(u) = \lambda_2(u) = 1$ then $\lambda(u) = 1$; if $\lambda_1(u) = \lambda_2(u) = \hat{0}$ then $\lambda(u) = \hat{0}$; and if $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$ then $\lambda(u) = 0$.
4. For $u \in X_4$, $\lambda_1(u) = \lambda_2(u) = 1$, or $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$.

For a coloring λ of S_e formed from λ_1 and λ_2, the minimum dominating set $D_e(\lambda)$ is the minimum set among the sets of $D_{e_1}(\lambda_1) \cup D_{e_2}(\lambda_2)$. For $e = \{z, r\}$, a minimum set $D_e(\lambda)$ among all colorings λ of S_e is a minimum dominating set of H. Notice that the original description of FT Algorithm for Step III in [56] does not have the part for handling the vertices colored black in data reduction process. We have added this part and our description is slightly different from the original one.

In the implementation of FT Algorithm, we put $a_{e_1}(\lambda_1)$ and a pointer to $D_{e_1}(\lambda_1)$ in a table T_1. Similarly, we put $a_{e_2}(\lambda_2)$ in a table T_2. Table T_1 has at most $3^{|S_{e_1}|} = 3^{|X_1|+|X_3|+|X_4|}$ entries and Table T_2 has at most $3^{|S_{e_2}|} = 3^{|X_2|+|X_3|+|X_4|}$ entries. We use the following index for the entries of T_1 and T_2: The entries of T_1 are first partitioned into $3^{|X_1|}$ groups by the

colors of the vertices in X_1. Similarly the entries of T_2 are partitioned into $3^{|X_2|}$ groups. The entries of each table within each group is further identified by the colors of the vertices in $X_3 \cup X_4$. To find a minimum $D_e(\lambda)$ from $D_{e_1}(\lambda_1)$ and $D_{e_2}(\lambda_2)$, we first compute

$$a_e(\lambda) = \min_{\lambda_1,\lambda_2 form \lambda} \{a_{e_1}(\lambda_1) + a_{e_2}(\lambda_2) - \#_1(X_3 \cup X_4, \lambda_1)\}$$

where $\#_1(X_3 \cup X_4, \lambda_1)$ is the number of vertices in $X_3 \cup X_4$ taking color 1 in λ_1 . The colors of λ_1 's and λ_2 's which form λ are the entries in the group of T_1 and the entries in the group of T_2 identified by the colors of λ for vertices of X_1 and X_2, respectively. Then, a corresponding minimum $D_e(\lambda)$ is computed. The results of $a_e(\lambda)$ are kept in a table T of at most $3^{|S_e|} = 3^{|X_1|+|X_2|+|X_3|}$ entries. The memory space required for computing table T and $D_e(\lambda)$ is $O((3^{|S_e|} + 3^{|S_{e_1}|} + 3^{|S_{e_2}|})k) = O(3^{\mathrm{bw}(H)}k)$ because $\max\{|S_e|, |S_{e_1}|, |S_{e_2}|\} \leq \mathrm{bw}(H)$ and $\max |D_e(\lambda)|, |D_{e_1}(\lambda_1)|, |D_{e_2}(\lambda_2)| \leq |V(H)| = O(k)$. If we only compute $\gamma(H)$ then we only need to compute table T and the required memory space is $O(3^{\mathrm{bw}(H)})$. Since there are $O(k)$ links in the branch-decomposition T_B, the total memory space required in Step III is $O(3^{\mathrm{bw}(H)}k^2)$ for computing a minimum dominating set and $O(3^{\mathrm{bw}(H)}k)$ for computing the dominating number of H. We call the above index method.

Dorn proposes using distance product of matrices to compute the minimum $D_e(\lambda)$ for all colorings λ [40]. We also implemented this approach (distance product method): The entries of Table T_1 are arranged into $r = 3^{|X_3|}$ matrices $A_1, ..., A_r$ of $3^{|X_1|}$ rows and $3^{|X_4|}$ columns ($|X_3| \leq 2$ for a planar graph with a fixed embedding and a branch-decomposition found by the algorithms used in this paper [40]). The entries of Table T_2 are arranged into r matrices $B_1, ..., B_r$ of $3^{|X_4|}$ rows and $3^{|X_2|}$ columns. Each row of $A_l (1 \leq l \leq r)$ is identified by a sequence of $|X_1|$ colors and each column of B_l is identified by a sequence of $|X_2|$ colors, with each color from $\{0, \hat{0}, 1\}$. Each column of A_l (each row of B_l) is identified by a sequence of $|X_4|$ colors. We arrange the columns of A_l in the increasing alphabetic order, defined by $\hat{0} < 0 < 1$, of the color sequences. We arrange the rows of B_l in the increasing alphabetic order, defined by $\hat{0} < 0 < 1$, of the color sequences. We define a one-to-one mapping between the colorings of $\{\hat{0}, 0, 1\}^{|X3|}$ and $1, 2, ..., r$ based on the alphabetic order defined by $\hat{0} < 0 < 1$. The value $a_{e_1}(\lambda_1)$ in each element of $A_l (1 \leq l \leq r)$ is changed to $a_{e_1}(\lambda_1) - \#_1(X_3 \cup X_4, \lambda_1)$. For every l with $1 \leq l \leq r$, the distance product $C_l = A_l \times B_l$ is computed, where C_l is a matrix of $|X_1|$ rows and $|X_2|$ columns, and $C_l[i, j] = \min\{A_l[i, k] + B_l[k, j], 1 \leq k \leq 3^{|X_4|}\}$. We say a coloring l of $\{0, \hat{0}, 1\}^{|X_3|}$ is formed by colorings l_1 and l_2 of $\{0, \hat{0}, 1\}^{|X_3|}$ if

- For $u \in X_3$, if $l_1(u) = l_2(u) = 1$ then $l(u) = 1$; if $l_1(u) = l_2(u) = \hat{0}$ then $l(u) = \hat{0}$; and

if $l_1(u) = 0$ and $l_2(u) = \hat{0}$, or $l_1(u) = \hat{0}$ and $l_2(u) = 0$ then $l(u) = 0$.

Every element $C_l[i, j]$ is further updated by $\min\{C_{l_1}[i, j], ..., C_{l_p}[i, j]\}$, where $l_1, ..., l_p$ are the colorings of $\{0, \hat{0}, 1\}^{|X_3|}$ which form coloring l. If a conventional $O(n^3)$ time algorithm is used for the distance product of the matrices, Step III takes $(2^{(3\log_4^3)\mathrm{bw}(H)}k) = O(2^{15.13\sqrt{k'}}k)$ time, and requires $O(3^{\mathrm{bw}(H)}k^2)$ and $O(3^{\mathrm{bw}(H)}k)$ memory spaces for computing a minimum dominating set and $\gamma(H)$, respectively, the same as those of the index method. If the distance product of matrices is realized by the $O(n^\omega)$ ($\omega < 2.376$) time fast matrix multiplication method, Step III has time complexity $O(2^{11.98\sqrt{k'}}n^{O(1)})$. In practice, the fast matrix multiplication method is slower due to the big hidden constant behind the Big-Oh than the conventional distance product method. The fast matrix multiplication method also requires more memory space due to the recursive computation.

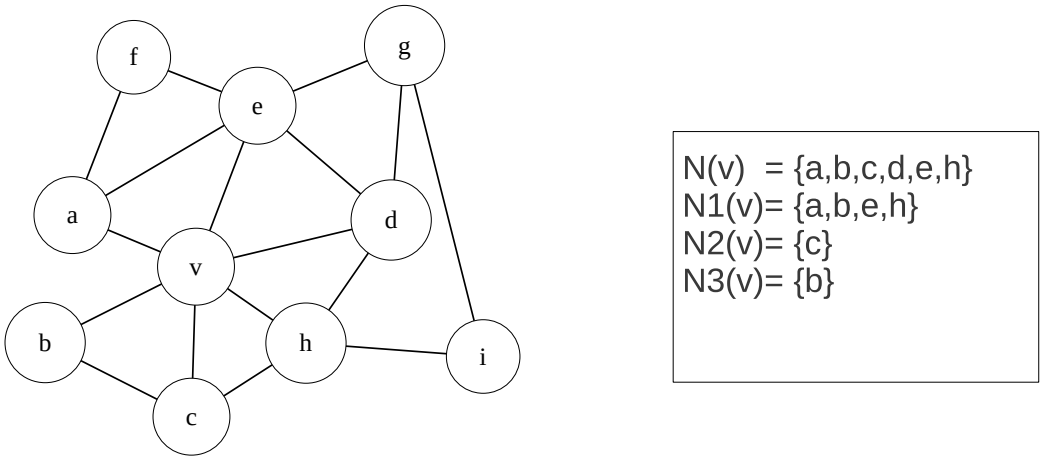


Figure 4.1: An example of N_1, N_2, N_3 of node v in the left side graph

4.2 Data reduction

In this section, we introduce the data reduction rules used in our implementation of FT Algorithm for Step I. All reduction rules of [7, 3] are used. To enhance the data reduction effect, we also propose some new reduction rules. Following the convention of FT Algorithm, we color each vertex of G by black or grey, and may remove some vertices from G by those reduction rules. After the data reduction step, we get a kernel $H(B \cup C, E)$, recall that B and C are the sets of black and grey vertices, respectively. For a vertex v, let

$N(v) = \{u|u, v \in E(G)\}$, $N[v] = N(v) \cup \{v\}, B(v) = B \cap N(v)$, and $C(v) = C \cap N(v)$. For a set U of vertices, let $N(U) = \bigcup_{v \in U} N(v)$. For a vertex u, if there is a black vertex $v \in N[u]$, we mark u dominated. Initially, every vertex of G is unmarked. In the data reduction step, some vertices are marked. Let X be the set of marked vertices and Y be the set of unmarked vertices. For $v \in V(G)$, the following is introduced in [7] (Figure 4.1 shows and example):

$$N_1(v) = B(v) \cup \{u|u \in C(v), N(u) \setminus N[v] \neq \emptyset\},$$

$$N_2(v) = \{u|u \in N(v) \setminus N_1(v), N(u) \cap N_1(v) \neq \emptyset\}, \text{ and}$$

$$N_3(v) = N(v) \setminus (N_1(v) \cup N_2(v)).$$

Rule 1 [7]. For $v \in V(G)$, if $N_3(v) \cap Y \neq \emptyset$ then remove $N_2(v)$ and $N_3(v)$ from G, color v black, and mark $N[v]$ dominated.

For a pair of vertices $v, w \in V(G)$, let $N(v, w) = N(v) \cup N(w) \setminus \{v, w\}$, $B(v, w) = B \cap N(v, w), C(v, w) = C \cap N(v, w)$, and $N[v, w] = N[v] \cup N[w]$. The following is introduced in [7](Figure 4.2 shows an example):

$$N_1(v, w) = B(v, w) \cup \{u|u \in C(v, w), N(u) \setminus N[v, w] \neq \emptyset\},$$

$$N_2(v, w) = \{u|u \in N(v, w) \setminus N_1(v, w), N(u) \cap N_1(v, w) \neq \emptyset\},$$

$$N_3(v, w) = N(v, w) \setminus (N_1(v, w) \cup N_2(v, w)).$$

Rule 2 [7]. For $v, w \in V(G)$ with both v and w grey, assume that $|N_3(v, w) \cap Y| \geq 2$ and $N_3(v, w) \cap Y$ can not be dominated by a single vertex of $N_2(v, w) \cup N_3(v, w)$.

Case 1: $N_3(v, w) \cap Y$ can be dominated by a single vertex of $\{v, w\}$.

- (1.1) If $N_3(v, w) \cap Y \subseteq N(v)$ and $N_3(v, w) \cap Y \subseteq N(w)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(v) \cap N(w)$ from G and add new gadget vertices z and z' with edges $\{v, z\}, \{w, z\}, \{v, z'\}$, and $\{w, z'\}$ to G.
- (1.2) If $N_3(v, w) \cap Y \subseteq N(v)$ but $N_3(v, w) \cap Y \not\subseteq N(w)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(v)$ from G, color v black, and mark $N[v]$ dominated.
- (1.3) If $N_3(v, w) \cap Y \subseteq N(w)$ but $N_3(v, w) \cap Y \not\subseteq N(v)$ then remove $N_3(v, w)$ and $N_2(v, w) \cap N(w)$ from G, color w black, and mark $N[w]$ dominated.

Case 2: If $N_3(v, w) \cap Y$ can not be dominated by a single vertex of $\{v, w\}$ then remove $N_2(v, w)$ and $N_3(v, w)$ from G, mark v and w black, and mark $N[v, w]$ dominated.

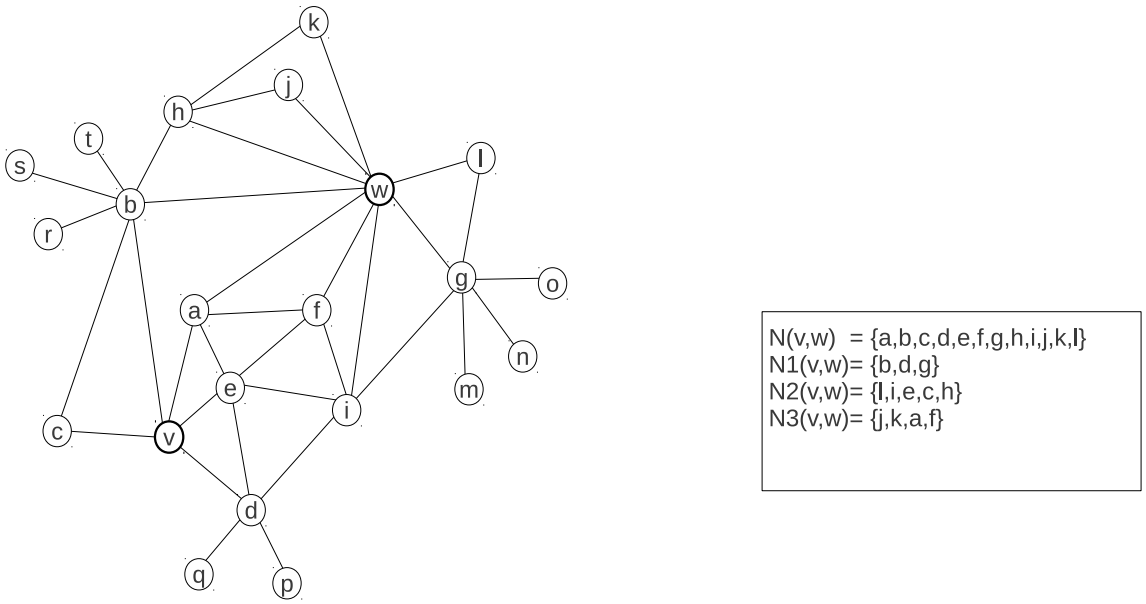


Figure 4.2: An example of N_1, N_2, N_3 of nodes v, w in the left side graph

In Rule 1 and Rule 2 (Cases 1.2, 1.3, and 2) of [7], gadget vertices are used to guarantee some vertices to be included in the solution set. In [3] the rules are implemented in a way that the vertices to be included in the solution set are removed. Our descriptions are slightly different from the previous ones: we do not use gadget vertices nor remove the vertices to be included to the solution set but color them black. Our descriptions allow us to have new reduction rules given below that may further reduce the size of the kernel.

Rule 3.

3.1: For $v, w \in V(G)$ with v black and w grey, if $(N_3(v,w) \cap Y) \setminus N(v) \neq \emptyset$ then remove $N_2(v,w) \cup N_3(v,w)$, color w black, and mark $N[w]$ dominated; otherwise remove $(N_2(v,w) \cup N_3(v,w)) \cap N(v)$.

3.2: For $v, w \in V(G)$ with v grey and w black, if $(N_3(v,w) \cap Y) \setminus N(w) \neq \emptyset$ then remove $N_2(v,w) \cup N_3(v,w)$, color v black, and mark $N[v]$ dominated; otherwise remove $(N_2(v,w) \cup N_3(v,w)) \cap N(w)$.

3.3: For $v, w \in V(G)$ with both v and w black, remove $N_2(v,w) \cup N_3(v,w)$.

Lemma 4.2.1 *Given a graph G, let G' be the graph obtained by applying Rule 3 for $v, w \in V(G)$. Then $size(G') \leq size(G)$, $\gamma(G') \leq \gamma(G)$, and a minimum dominating set D' of G' that contains all black vertices of G' is a minimum dominating set of G that contains all black vertices of G.*

Proof: For $v, w \in V(G)$ with v black and w grey, assume that $(N_3(v, w) \cap Y) \setminus N(v) \neq \emptyset$. For $u \in (N_3(v, w) \cap Y) \setminus N(v)$ and x which dominates $u, x \in \{w\} \cup N_2(v, w) \cup N_3(v, w)$. Since $N(N_2(v, w) \cup N_3(v, w)) \subseteq N[v] \cup N[w]$, we should include w into D to dominate $(N_3(v, w) \cap Y) \setminus N(v)$. Therefore, we can remove $N_2(v, w) \cup N_3(v, w)$ from G. Assume that $(N_3(v, w) \cap Y) \setminus N(v) = \emptyset$. For $u \in (N_2(v, w) \cup N_3(v, w)) \cap N(v)$, u is dominated by v and $N(v) \cup N(u) \subseteq N(v) \cup N(w)$. This implies that we can at least include w rather than u to get D. At this point, we can not decide if we should include w into D or not because there might be a vertex x with $N(w) \subseteq N(x)$ that should be included in D. But we can exclude $(N2(v, w) \cup N3(v, w)) \cap N(v)$ from D. Since $(N_2(v, w) \cup N_3(v, w)) \cap N(v)$ is dominated by v, we can remove $(N_2(v, w) \cup N3(v, w)) \cap N(v)$ from G. This completes the proof for (3.1). The proof for (3.2) is a symmetric argument of that for (3.1). For $v, w \in V(G)$ with both v and w black, since $N(N_2(v, w) \cup N_3(v, w)) \subseteq N[v] \cup N[w]$, we can remove $N_2(v, w) \cup N_3(v, w)$ from G. []

Rule 4 [7]

4.1: Delete edges between vertices of X (vertices marked dominated).

4.2: If $u \in X$ has $|C(u)| \leq 1$ then remove u.

4.3: For $u \in X$ with $C(u) \cap Y = \{u_1, u_2\}$, if u_1 and u_2 are connected by a path of length at most 2 then remove u.

4.4: For $u \in X$ with $C(u) \cap Y = \{u_1, u_2, u_3\}$, if $\{u_1, u_2\}, \{u_2, u_3\} \in E(G)$ then remove u.

To perform the data reduction, we first apply Rule 1 for every vertex of G. Next for every pair of vertices v and w of G, we apply either Rule 2 or Rule 3 depending on the colors of v and w. Then we apply Rule 4. We repeat the above until Rules 1 to 4 do not change the graph. From the results of [7, 3] on Rules 1, 2, and 4, and Lemma 4.2.1, we have the following result.

Theorem 4.2.1 *Given a planar graph G, let $H(B \cup C, E)$ be the kernel obtained by applying the reduction rules described above and D' be a minimum vertex set of $H(B \cup C, E)$ such that $D' \supseteq B$ and D' dominates C. Then a minimum dominating set D of G can be constructed from D' in linear time.*

In our implementation of FT Algorithm, $D' = D$. Since the vertices of B are included to D' in Step I, the number of vertices to be included in D' in Step III is $|D| - |B|$. Therefore, the size of the dominating set for the kernel decided in Step III is actually $k' = \gamma(G) - |B|$. If Step I gives an non-empty set B of black vertices, k' is smaller than $k = \gamma(G)$. Given a planar graph G, let $H(B \cup C, E)$ be the kernel obtained from Step I, T_B be an optimal branch-decomposition of H, and $l(H) = \max\{|C \cap S_e|, e \in E(T_B)\}$. It is shown in [7] that $H(B \cup C, E)$ can be computed in $O(n^3)$ time. T_B can be computed by either the edge-contraction algorithm [61] or a divide-and-conquer algorithm [18] in $O(|E(H)|^3)$ time. It is shown in [56] that Step III has time complexity $O(2^{(3\log\frac{3}{4})l(H)}|E(H)|)$. Therefore, FT Algorithm takes $O(2^{(3\log\frac{3}{4})l(H)}|E(H)|+n^3)$ time to solve the PLANAR DOMINATING SET problem. Notice that $l(H) \leq \mathrm{bw}(H)$ and in what follows, we use $l(H)$ for the branchwidth of kernel H.

4.3 Computational results

We implemented FT Algorithm and tested our implementations on six classes of planar graphs from some libraries including LEDA [1, 88] and PIGALE [2]. LEDA generates two types of planar graphs. One type of graphs are the random maximal planar graphs and their subgraphs and the other type of graphs are the planar graphs based on some geometric properties, including the Delaunay triangulations and triangulations of points, and the intersection graphs of segments, uniformly distributed in a two-dimensional plane. Instances of Class (1) are the random maximal graphs and their subgraphs generated by LEDA. This class of instances has been used by Alber et al. in their studies on the data reduction rules used in Step I [3, 7] and the tree-decomposition based subexponential algorithms for the vertex cover and dominating set problems [4, 6]. Instances of Class (2) are Delaunay triangulations of point sets taken from TSPLIB [95]. Instances of Classes (3) and (4) are the triangulations and intersection graphs generated by LEDA, respectively. Instances of Class (5) are Gabriel graphs of the points uniformly distributed in a two-dimensional plane.

Instances of Classes (2)-(5) are graphs based on some geometric properties. The DOMINATING SET problem on those graphs has important applications such as the virtual backbone design of wireless networks [79]. Instances of Class (6) are random planar graphs generated by the PIGALE library [2]. PIGALE provides a number of planar graph generators. We used a function in the PIGALE library that randomly generates one of all possible 2-connected planar graphs with a given number of edges based on the algorithms of [101]

In addition to the tested classes of planar graphs in Chapter 3, we study the performance of FT Algorithm on three additional classes of planar graphs including maximal planar graphs, Delauny triangulation graphs generated by LEDA, and Gabriel graphs. The branchwidth of maximal planar graphs is at most 4 (see Appendix A) which makes them an interesting class of graphs in our study. This type of planar graphs is also studied for many NP-hard problems in planar graphs, such as PLANAR DOMINATING SET probelm [3], face labelling [28] and integer-magic spectra [78]. Alber et al. [3] proposed reduction rules for linear size kernel of planar dominating set problem. They also have a computational study paper on their reduction rules. Maximal planar graphs are used in the paper. The notion of triangulated graphs applies to problem solving within as widely different areas as solution of sparse symmetric systems of linear equations, pedigree analysis, and evidence propagation in belief graphs. The Gabriel graph is one of widely used geometric structures for topology control in wireless ad hoc networks. Many problems in wireless ad hoc networks are solved based on modelling the network using this class of graphs [73, 80, 14].

Step I of FT Algorithm is implemented as described in the previous section. To compute an optimal branch-decomposition T_B, we use the divide-and-conquer algorithm [18]. For Step III, both the index and distance product methods are used. To save memory, we compute the colorings λ and sets $D_e(\lambda)$ for each link e of T_B in the postorder. Once the colorings λ and sets $D_e(\lambda)$ are computed for a link e, the solutions for the child links of e are discarded. The computer used for testing has an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ (2.4 GHz) and 3 Gbyte memory. The operating system is SUSE Linux 10.2 and the programming language used is C++.

We report the computational results for finding both $\gamma(G)$ and a minimum dominating set of G by FT Algorithm with the index method in Step III in Table 4.1. For Step I, we give the number $|B|$ of vertices of an optimal dominating set decided in the data reduction and the running time of the step. For Step II, we give the size $|E(H)|$ and branchwidth $l(H) = \max\{|C \cap S_e|, e \in E(T_B)\}$ of kernel H, and the running time of the step. For Step

III, we give the dominating number $\gamma(G)$ obtained by FT Algorithm and the running time of the step. The running time is in seconds, and Steps I, II, and III have time complexities $O(|E(G)|^3), O(|E(H)|^3)$, and $O(2^{(3log_4^3)l(H)}|E(H)|)$, respectively. We use the number of edges to express the size of an instance or a kernel.

For each instance size, we have tested three graphs of similar size for Classes (1) and (3)-(6), and Table 4.1 contains the graph with the worst case running time. Notice that multiple graphs of similar size in Class (2) are not available. The average performance of FT Algorithm over three graphs for some large instances is given in Table 4.3. We also report the running time of FT Algorithm for computing $\gamma(G)$ only for some large instances in Table 4.1. For those instances (marked with an $*$) FT Algorithm can not compute a minimum dominating set by the computer used in this study because it requires more than 3 GByte memory space but can compute the $\gamma(G)$.

As shown in the Appendix A, the instances of Class (1) have branchwidth at most four. These instances have small kernels and Step I is very effective. For the instances included in the table, $|B|$ is very close to $\gamma(G)$ (i.e., Step I finds most vertices in an optimal dominating set) and the kernels are much smaller than the original instances. For some smaller instances not reported in the table, Step I already finds optimal dominating sets. Because the kernels have small size and branchwidth, FT Algorithm is efficient for the instances in this class. For example, an optimal dominating set can be computed for large instances of size up to about 40,000 edges in about 20 min. It is reported in [4] that instances of size about 6000 edges can be solved in about 30 min by a tree-decomposition based algorithm on a computer with a CPU of 750 MHz and 720 MBytes memory space. These results suggest that FT Algorithm is more efficient than the algorithm used in [4] for the graphs in this class.

For Classes (2) and (5), the branchwidth of instances increases fast in instance size (e.g., Class (2) instances rd400 of 1183 edges and u2152 of 6312 edges have branchwidth 17 and 31, respectively, Class (5) instances Gab800 of 1533 edges and Gab2000 of 3911 edges have branchwidth 16 and 26, respectively). For the instances tested, the kernel H of an instance G has the same branchwidth and same size as or only slightly smaller than those of G. The computation time increases significantly when the branchwidth of the kernels increases. This coincides with the theoretical time complexity of FT Algorithm which runs exponentially in $l(H)$. For Classes (3) and (4), the branchwidth of instances increases slowly in instance size. The data reduction is effective for instances in these classes. For most instances, the kernel size is at most half of the instance size and the branchwidth of the kernel is usually

smaller than that of the instance as well. Our data show that the minimum dominating set can be found for instances of size up to about thirty thousand edges in a practical time and memory space. For large instances, the size $|E(H)|$ of kernel H is also important to the running time of Step III. For example, FT Algorithm takes more time to solve Instance *rand15000* than that for *rand10000*. The time difference comes from the differences of both $l(H)$ and $|E(H)|$. For Class (6), the branchwidth of instances does not grow in the instance size. FT Algorithm is efficient for the instances in this class. For instances of large size and small branchwidth, Step III may not dominant the running time. Our results show that the $O(n^3)$ time data reduction and branch-decomposition finding take more time than the dynamic programming part for those instances.

Table 4.1 only contains the instances well scaled within some size ranges. We have tested FT Algorithm on graphs with size different from those in 4.1. The results are similar to those in the table, the running time mainly depends on $l(H)$ and then $|E(H)|$.

FT Algorithm requires in Step III $O(2^{10.1l(H)}k^2)$ and $O^(2^{10.1l(H)}k)$ memory spaces for computing a minimum dominating set and $\gamma(H)$ of kernel H, respectively. The memory requirement seems a bottleneck for solving instances with large branchwidth. We report the memory space (in MBytes) used by FT Algorithm in Table 4.2 for large instances in Classes (1)-(6). Our data show that FT Algorithm can compute a minimum dominating set and the dominating number for instances with the branchwidth of kernels at most 13 ($l(H) \leq 13$) and at most 14 ($l(H) \leq 14$), respectively, by 3 GBytes memory space. The average memory space over three graphs used by FT Algorithm for some large instances is given in Table4.3.

Our computational results confirm the theoretical analysis of FT Algorithm: It is efficient for graphs with small branchwidth but time and memory consuming for graphs with large branchwidth. This suggests that the branchwidth of a planar graph is a key parameter to decide if a problem on the graph can be solved efficiently or not. For example, Class (1) graphs have branchwidth at most four and thus admit efficient algorithms for many hard problems. On the other hand, the problems on graphs in Classes (2) and (5) are less tractable because these graphs have large branchwidth.

Both the theoretical analysis and computational study suggest that computing a kernel H with smaller $l(H)$ and $|E(H)|$ is a most effective way to improve the efficiency of FT Algorithm. For this purpose, we proposed new reduction rules (Rule 3). Recall that H is the kernel obtained by new reduction rules (Rules 1, 2, 3, and 4) and let H' be the

Class	Graph	$\|E(G)\|$	$bw(G)$	Step I		Step II			Step III		total
	G			$\|B\|$	time	$\|E(H)\|$	$bw(H)$	time	$\gamma_c(G)$	time	time
(1)	max1500	3860	4	228	12	78	3	< 1	236	< 1	12
	max6000	7480	4	2214	55	32	2	< 1	2219	< 1	55
	max8000	13395	4	2186	336	194	3	< 1	2211	< 1	337
	max11000	28537	4	1679	799	208	4	1	1695	< 1	800
	max13500	38067	4	1758	1203	302	3	1	1779	< 1	1204
(2)	kroB150	436	10	0	< 1	436	10	< 1	23	10	10
	pr226	586	7	12	1	126	6	< 1	21	< 1	1
	pr299	864	11	1	< 1	824	11	1	47	35	37
	tsp225	622	12	0	< 1	622	12	1	37	109	110
	a280	788	13	1	< 1	730	13	1	43	336	337
(3)	tri2000	5977	8	136	57	3192	7	140	321	1	198
	tri4000	11969	9	252	256	6888	7	1641	653	6	1903
	tri6000	17979	9	312	566	11691	8	2991	975	19	3576
	tri8000	23975	9	497	830	13524	7	6900	1283	20	7750
	tri10000	29976	9	605	1434	17298	7	15028	1606	33	16495
(4)	rand3000	4928	9	554	21	1918	6	8	823	1	30
	rand6000	10293	11	836	95	5598	9	25	1563	30	150
	rand10000	17578	13	1192	376	10706	10	381	2535	112	869
	rand15000	26717	14	1570	875	17810	12	354	3758	1540	2769
	rand16000*	28624	13	1612	826	19700	13	2063	4002*	3028	5917
	rand20000*	35975	14	1993	1904	24786	14	3632	4963*	8457	13993
(5)	Gab100	182	7	3	< 1	162	7	< 1	24	1	1
	Gab300	552	10	5	< 1	516	10	1	70	23	25
	Gab500	949	13	4	1	919	12	56	115	181	238
	Gab600*	1174	14	11	2	1097	14	5	135*	3067	3074
	Gab700*	1302	14	8	1	1255	14	9	162*	5700	5710
(6)	P1277	2128	9	116	9	1353	9	13	323	2	24
	P2518	4266	9	329	32	1876	5	27	621	1	60
	P4206	7124	8	513	92	3543	6	16	1057	2	110
	P5995	10082	8	738	188	4920	5	20	1495	1	209
	P7595	12788	7	965	336	5908	6	18	1903	< 1	354

Table 4.1: Computational results (time in seconds) of FT Algorithm with the index method in Step III for instances of Classes (1)(6). For the instances marked with *, the time is for computing the dominating number only because the 3 GByte memory is not enough for computing a minimum dominating set.

Class	Instance	$\|E(G)\|$	$bw(G)$	$l(H)$	Memory(MByte)
(1)	max11000	28537	4	4	480
	max13500	38067	4	3	720
(2)	a280	788	13	13	510
	rd400	1183	17	17	> 300
(3)	tri8000	23975	9	7	710
	tri10000	29976	9	7	1210(StepIII,1030)
(4)	rand10000	17578	13	10	470
	rand15000	26717	14	12	1800
	rand16000*	28624	13	13	660
	rand20000*	35975	14	14	1200
	rand25000*	45278	15	15	> 3000
(5)	Gab300	552	10	10	40
	Gab500	949	13	12	660
	Gab700*	1302	14	14	1200
	Gab800*	1533	16	16	> 3000
(6)	P5995	10082	8	5	150
	P7595	12788	7	6	240

Table 4.2: Memory space (in MBytes) of FT Algorithm with the index method in Step III for instances of Classes (1)(6). For the instances marked with *, the memory space is for computing the dominating number only. X indicates that the problem can not be solved for the instance because it requires more than 3 Gbytes memory space.

Class	Graph	Average $\|E(G)\|$	Average $bw(G)$	Average $l(H)$	Average $\gamma(G)$	Average time	Worst time	Average memory	Worst memory
(1)	max13500	38322	4	3	1763	1011	1204	720	720
(3)	tri10000	29973	9	7	1609	13342	16495	1200	1360
(4)	rand15000	26706	15	11	3764	2040	2769	1190	1800
(5)	Gab500	945	13	12	117	200	238	670	710
(6)	P7595	12760	7	6	1894	316	353	240	240

Table 4.3: Average performance of FT Algorithm over three graphs for each instance size. The time is in seconds and memory is in Mbytes.

Class	Graph G	$\|E(G)\|$	$bw(G)$	Results without new rules				Results with new rules			
				$\|B'\|$	$\|E(H')\|$	$l(H')$	Time	$\|B\|$	$\|E(H)\|$	$l(H)$	Time
(1)	max1500	3860	4	225	118	3	12	228	78	3	12
	max6000	7480	4	2212	41	2	58	2214	32	2	55
	max8000	13395	4	2183	218	3	353	2186	194	3	337
	max11000	28537	4	1671	287	4	892	1679	208	4	800
	max13500	38067	4	1752	362	4	1291	1758	302	3	1204
(3)	tri2000	5977	8	102	3787	7	353	136	3192	7	198
	tri4000	11969	9	214	7541	7	1941	252	6888	7	1903
	tri6000	17979	9	277	12370	8	3895	312	11691	8	3576
	tri8000	23975	9	421	14953	8	10192	497	13524	7	7750
	tri10000	29976	9	551	18273	8	18945	605	17298	7	16495
(4)	rand3000	4928	9	545	1987	6	30	554	1918	6	30
	rand6000	10293	11	832	5675	9	154	836	5598	9	150
	rand10000	17578	13	1176	10861	11	892	1192	10706	10	869
	rand13000	22953	13	1454	14856	10	1662	1589	14646	10	1169
	rand15000	26717	14	1553	17984	12	2834	1570	17810	12	2769
(6)	P1277	2128	9	112	1371	9	41	116	1353	9	24
	P2518	4266	9	291	2139	6	69	329	1876	5	60
	P4206	7124	8	478	3780	6	116	513	3543	6	110
	P5995	10082	8	671	5372	5	224	738	4920	5	209
	P7595	12788	7	917	6231	6	363	965	5908	6	354

Table 4.4: The results (time in seconds) of using new data reduction rules and without using the new rules in Step I.

kernel obtained by applying only the previous known reduction rules (Rules 1, 2, and 4). Since all nodes colored black (resp. nodes deleted) by previous rules are also colored balck (resp. deleted) by new rules, $l(H) \leq l(H')$ and $|E(H)| \leq |E(H')|$. For Classes (2) and (5), $l(H) = l(H') = \text{bw}(G)$ and $|E(H)| = |E(H')| = |E(G)|$ for most instances, that is, the effect of data reduction is very limited. However, for instances in other classes, data reduction is effective and our new rules improve the efficiency of FT Algorithm. For instances of Classes (1), (3), (4), and (6), Table 4.4 shows the computational results of FT Algorithm when previous rules and new rules are used. In the table, t_{old} and t_{new} (resp. $|B'|$ and $|B|$) are the total running times (resp. the numbers of vertices in an optimal dominating set decided in Step I) when previous rules and new rules are used, respectively. The data show that $l(H) = l(H')$ and $|E(H)| < |E(H')|$ for most instances. The total running time is improved when new rules are used: $t_{new} < t_{old}$ for most instances in the table. The improvement is instance dependent and t_{new}/t_{old} varies from 56% to 100%. The average of t_{new}/t_{old} over the five instances of Class (1) is about 95%. Similarly, the averages of t_{new}/t_{old} for Classes (3), (4), and (6) are about $80\%, 90\%$, and 85%, respectively. The improvement of the total running time is obtained mainly from Step III. The running time of Step I when new rules

are used is about the same as that when previous rules are used (instance dependent).

Step III of FT Algorithm can also be realized by the distance product method proposed by Dorn [40]. When a conventional $O(n^3)$ time method is used to realize the distance product of matrices, the distance product method has the same time complexity as that of the index method. Theoretically, using the fast matrix multiplication for the distance product of integer matrices [111] can reduce the order of time complexity. In practice, using the fast matrix multiplication (e.g., the Strassens method) for distance product of matrices is slower than the conventional method. We report in Table 4.5 the running times of Step III by the index method and the distance product method (with conventional distance product). Our data show that the running times of the two methods are similar. Both methods require a similar size of memory space as well.

Class	Graph	$\|E(G)\|$	Distance product time	Index method time
(1)	max8000	13395	< 1	< 1
	max11000	28537	2	< 1
	max13500	38067	< 1	< 1
(2)	pr299	864	25	35
	tsp225	622	104	109
	a280	778	310	336
(3)	tri5000	14969	56	7
	tri6000	17979	62	11
	tri7000	20980	163	14
(4)	rand5000	8451	12	2
	rand6000	10293	21	30
	rand8000	13816	83	38
(5)	Gab00	182	1	1
	Gab200	366	1	2
	Gab300	552	18	23
(6)	P1277	2128	3	2
	P5995	10092	46	3
	P7595	12691	5	1

Table 4.5: The results (time in seconds) of using distance product method and index method in Step III.

Chapter 5

CONNECTED DOMINATING SET problem in planar graphs

5.1 Algorithm for planar CDS problem

Recently, significant progress has been made on the fixed-parameter algorithms for the PLANAR DOMINATING SET problem [57, 40] and practical performance of those algorithms have been reported in [87]. The notions of tree/branch-decompositions introduced by Robertson and Seymour [96, 97, 98] play a central role in those algorithms. Although the DOMINATING SET problem and the CDS problem are closely related, they have different properties from the tree/branch-decomposition based algorithm point of view. In particular, the techniques used to solve the DOMINATING SET problem do not seem to work for the CDS problem. One of the main reasons of such discrepancy is that *connectivity* is a *non-local property*. In the tree/branch-decomposition based approach, the input graph is partitioned into subgraphs by a tree/branch-decomposition of the graph; then partial solutions are worked out by enumeration for each minimal subgraph and the partial solutions of subgraphs are merged into partial solutions of a larger subgraph until the solution of the entire graph is found. Notice that each subgraph is separated from the rest of the input graph by a vertex-cut set. We call this vertex-cut set the *boundary* of the subgraph. Informally, the structure of a partial solution at the boundary of a subgraph and that in the entire subgraph are called the *local structure* and *non-local structure* of the partial solution, respectively. A problem is called *local* if the merge steps can be done by only looking at

the local structure of the partial solutions. A problem is called *non-local* if the non-local structures need to be checked in the merge step. The DOMINATING SET problem is local while the CDS problem is non-local. It is more difficult to perform the merge steps for non-local problems.

Along the lines to clear the hurdles caused by the non-local property, Dorn et al. [42, 43] propose a new technique to design sub-exponential time exact algorithms for the non-local problems in planar graphs. This new technique is based on the geometric properties of branch-decomposition of graphs with a planar embedding in a sphere and the properties of non-crossing partitions in the embedding. This new technique consists of two main steps. In the first step an optimal *sphere-cut decomposition* (a branch-decomposition with a desired geometric property, see Chapter 2 for definition) for an input planar graph G is constructed. This can be computed in $O(n^3)$ time [102, 61]. In the second step, the dynamic programming method based on the sphere-cut decomposition is applied to compute the partial solutions and merge these solutions to form the solution for the input graph. Based on this new technique, they show that many non-local problems in planar graphs can be solved in $2^{O(\sqrt{n})}$ time [42, 43]. Especially, it is shown that the PLANAR CDS problem can be solved in $O(2^{O(bw(G))}n+n^3)$ and $O(2^{9.822\sqrt{n}}n+n^3)$ time [43][1]. It is mentioned in [40] that the running time can be further improved to $O(2^{8.11\sqrt{n}}n+n^3)$ if the fast distance matrix multiplication is applied to the second step. The time bound $O(2^{O(\mathrm{bw}(G))}n+n^3)$ implies that the PLANAR CDS problem admits an $O(2^{O(\sqrt{\gamma_c(G)})}n+n^3)$ time fixed-parameter algorithm. It is known that the PLANAR CDS problem admits a linear size kernel [72, 86] and such a kernel can be computed in $O(n^3)$ time [72]. Applying the algorithm of [72] to shrink the input graph G into a linear size kernel, the DPBF Algorithm solves the PLANAR CDS problem in $O(2^{O(\sqrt{\gamma_c(G)})}\gamma_c(G)+n^3)$ time.

Because DPBF Algorithm is briefly introduced and the analysis is not explicitly given in [42, 43, 40], we give a detailed description of the algorithm and analyze its running time. By a more careful analysis, we show that a conventional version of DPBF Algorithm solves the PLANAR CDS problem in $O(2^{4.62\mathrm{bw}(G)}\gamma_c(G)+n^3)$ time. If the fast distance matrix multiplication is applied, the algorithm solves the PLANAR CDS problem in $O(2^{3.722\mathrm{bw}(G)}\gamma_c(G)+n^3)$ time. From $\mathrm{bw}(G) \leq \sqrt{4.5n}$ [56], we get $O(2^{9.8\sqrt{n}}n+n^3)$ and $O(2^{7.9\sqrt{n}}n+n^3)$ for the running time of DPBF Algorithm, respectively.

[1]The constant in $O(\mathrm{bw}(G))$ is not explicitly given in [42, 43]

It is known $\text{bw}(G) \leq 3\sqrt{4.5\gamma(G)}$ [57, 56]. Since $\gamma(G) \leq \gamma_c(G)$, DPBF Algorithm solves the planar CDS problem in $O(2^{23.7\sqrt{\gamma_c(G)}}\gamma_c(G) + n^3)$ time. We prove that $\text{bw}(G) \leq 2\sqrt{10\gamma_c(G)} + 32$ for a planar graph G. This improves the previous bound of $\text{bw}(G) \leq 3\sqrt{4.5\gamma_c(G)}$. From this result, the PLANAR CDS problem admits an $O(2^{23.54\sqrt{\gamma_c(G)}}\gamma_c(G) + n^3)$ time fixed-parameter algorithm.

5.1.1 Sphere-cut branch-decomposition based approach

The new technique by Dorn el al. for non-local planar problems is within the framework of the branch-decomposition based approach which has two major steps as shown below.

1. Compute a branch-decomposition T of the input graph.

2. Apply the dynamic programming method based on T to solve the problem:

   A link e of T is called a *leaf link* if e contains a leaf node of T, otherwise called an *internal link*. T is converted to a rooted binary tree by replacing a link $\{x, y\}$ of T with three links $\{x, z\}, \{y, z\}, \{z, r\}$, where z and r are new nodes to T, r is the root, and $\{z, r\}$ is an internal link. A link e' (resp. a node x) is called a *descendant link* (resp. *descendant node*) of link e if e is in the path from e' (resp. x) to the root r of T. For a link e of T, let $(A_e, \overline{A_e})$ be the separation induced by e with A_e the set of leaf nodes of T (set of edges of G) that are descendant nodes of e. For an optimization problem P, all possible partial solutions of P in the subgraph $G[A_e]$ are computed first, say by enumeration, for each leaf link e of T. For an internal link e of T, e has two descendant links e_1 and e_2 incident to e (e_1 and e_2 are called *child* links of e). Notice that $A_e = A_{e_1} \cup A_{e_2}$. Assume that all partial solutions of P in the subgraph $G[A_{e_1}]$ and those in $G[A_{e_2}]$ have been computed. Then all possible partial solutions in the subgraph $G[A_e]$ are computed by merging the partial solutions in $G[A_{e_1}]$ and those in $G[A_{e_2}]$. The merging process is performed in a bottom-up way, from leaf links to the link $\{z, r\}$, to find the solution of P in G. The merge process can be realized by the dynamic programming method with the partial solutions in $G[A_e]$ for each link e of T kept in a table.

For some problems such as the INDEPENDENT SET problem and DOMINATING SET problem, a partial solution in $G[A_e]$ can be identified by a fixed number of states of each vertex in $\partial(A_e)$. For example, a partial solution D of the DOMINATING SET problem

in $G[A_e]$ can be identified by three states of each vertex $u \in \partial(A_e)$: $u \in D$, $u \notin D$ but dominated by a vertex of D, and $u \notin D$ but u not dominated by any vertex of D. For such problems we can assign a fixed number of colors to each vertex of $\partial(A_e)$ such that every partial solution in $G[A_e]$ can be uniquely identified by a coloring of $\partial(A_e)$ and all partial solutions in $G[A_e]$ can be computed from the colorings of the vertices in $\partial(A_{e_1})$ and those of the vertices in $\partial(A_{e_2})$. In other words, to compute the partial solutions of $G[A_e]$, we can look at only the local structures of the partial solutions at $\partial(A_{e_1})$ and $\partial(A_{e_2})$. Because of these properties, the INDEPENDENT SET problem and the DOMINATING SET problem are known having a *local structure*.

For the CDS problem, however, the connectivity information in a partial solution in $G[A_e]$ may not be expressed by a fixed number of colors of each vertex of $\partial(A_e)$. In the merge step, the structures of the partial solutions in the entire subgraphs $G[A_{e_1}]$ and $G[A_{e_2}]$ may have to be checked. Because of this, the CDS problem is known having a *non-local structure*.

Dorn et al. give a new technique which makes the branch-decomposition based approach applicable to many problems with the non-local structure in planar graphs [42, 43]. This new technique is based on two observations. One is the geometric property of the sphere-cut decomposition T of plane graph G: For any link e of T and the separation $(A_e, \overline{A_e})$ induced by e, there is a noose ν_e such that ν_e induces $(A_e, \overline{A_e})$, ν_e partitions the sphere Σ into two regions, all edges of A_e are in one region, and all edges of $\overline{A_e}$ are in the other region. Notice that ν_e intersects all vertices of $\partial(A_e)$. The other observation is known as the non-crossing partitions: Let $P_1, ..., P_r$ be the subsets of A_e such that $G[P_i]$ is connected for each $1 \leq i \leq r$ and $G[P_i \cup P_j]$ is not connected for every pair of $1 \leq i \neq j \leq r$. We call $P_1, ..., P_r$ *disjoint components*. Two components P_i and P_j are called *crossing* if there are $u, u' \in V(P_i) \cap \partial(A_e)$ and $v, v' \in V(P_j) \cap \partial(A_e)$ such that the four vertices appear on ν_e in the orders u, v, u', v', otherwise *non-crossing*. Notice that if P_i and P_j are crossing then $G[P_i \cup P_j]$ is connected because $G[A_e]$ is a plane graph. So, any pair of disjoint components are non-crossing. The sphere-cut decomposition and the non-crossing partitions make it possible to compute the partial solutions in $G[A_e]$ by only looking at the local structures of partial solutions in $G[A_{e_1}]$ at $\partial(A_{e_1})$ and those in $G[A_{e_2}]$ at $\partial(A_{e_2})$.

For a minimum connected dominating set, D of G, the subgraph $G[D \cap V(A_e)]$ of $G[A_e]$ induced by D consists of disjoint components $P_1, ..., P_r$ with $|V(P_i) \cap \partial(A_e)| \geq 1$ for every $1 \leq i \leq r$. We assume the vertices of $\partial(A_e)$ are indexed as $u_1, u_2, ..., u_k$ in the clockwise order

as they appear in the noose ν_e. If $|V(P_i) \cap \partial(A_e)| \geq 2$, we call the vertex of $V(P_i) \cap \partial(A_e)$ with the smallest index the *small end*, the vertex of $V(P_i) \cap \partial(A_e)$ with the largest index the *large end* and other vertices of $V(P_i) \cap \partial(A_e)$ the *middle vertices* of P_i. From the geometric property of sphere-cut decomposition and the non-crossing partitions, $P_1, ..., P_r$ can be identified by six states of each vertex $u \in \partial(A_e)$ [43]:

1. u does not appear in any P_i and is dominated by some vertex of $D \cap V(A_e)$.
2. u does not appear in any P_i and is not dominated by any vertex of $D \cap V(A_e)$.
3. u is the small end of some P_i.
4. u is the large end of some P_i.
5. u is a middle vertex of some P_i.
6. u is the only vertex of some $V(P_i) \cap \partial(A_e)$.

5.1.2 Algorithm description

There are three major steps in the algorithm. Let G be a plane graph of n vertices.

Step I: Compute a kernel H of G with $|V(H)| = O(\gamma_c(G))$. This can be done in $O(n^3)$ time [72]. A plane graph H can be naturally obtained from the plane graph G by removing the vertices/edges deleted in the kernelization.

Step II: Compute a sphere-cut decomposition T of H with width $\mathrm{bw}(H)$. This can be done in $O((\gamma_c(H))^3)$ time [102, 61].

Step III: Compute a minimum connected dominating set D of H using the dynamic programming method based on T and compute a minimum connected dominating set of G from D.

The key issues in the dynamic programming of Step III are a coloring scheme for the vertices of $\partial(A_e)$ for each link e of T with two children e_1 and e_2 and the computation of the partial solutions in $H[A_e]$ from the colorings of $\partial(A_{e_1})$ and those of $\partial(A_{e_2})$.

Let D be a connected dominating set of H. Then $D \cap V(A_e)$ has the following properties:

1. For each vertex $u \in V(A_e) \setminus \partial(A_e)$ either $u \in D \cap V(A_e)$ or there is a vertex $v \in D \cap V(A_e)$ such that $\{u, v\} \in A_e$.

2. Each of the disjoint components of $H[D \cap V(A_e)]$ has at least one vertex in $\partial(A_e)$.

So, a partial solution of $H[A_e]$ is a subset of $V(A_e)$ having properties (1) and (2) with $D \cap V(A_e)$ replaced by the subset. To give a coloring scheme for the vertices of $\partial(A_e)$ for identifying a partial solution, we use three basic colors in planar DOMINATING SET problem introduced in Chapter 4.

- **Black:** denoted by 1, for a vertex which is included in the partial solution.
- **White:** denoted by 0, for a vertex which is dominated and not included in the partial solution.
- **Grey:** denoted by $\hat{0}$, for a vertex which has not been decided to have the color black or white at the current step.

Let $b = |\partial(A_e)|$, $b_1 = |\partial(A_{e_1})|$ and $b_2 = |\partial(A_{e_2})|$. A coloring $\lambda \in \{0, \hat{0}, 1\}^b$ is called a *basic-coloring* of $\partial(A_e)$. Let U be a partial solution of $H[A_e]$. Then each vertex of $U \cap \partial(A_e)$ is given a black color in a basic-coloring of $\partial(A_e)$ for identifying U. Each vertex of $\partial(A_e)$ with the black color appears in one of the disjoint components $P_1, ..., P_r$ of $H[U \cap V(A_e)]$. The basic color black is converted to four different colors as follows.

- $1_{\lfloor}$: for a vertex which is the small end of some P_i.
- $1_{\rfloor}$: for a vertex which is the large end of some P_i.
- 1^* : for a vertex which is a middle vertex of some P_i.
- $\hat{1}$: for a vertex which is the only vertex of some $V(P_i) \cap \partial(A_e)$.

Figure 5.1 gives an example on how we assign these colors to the vertices of $\partial(A_e)$. In this figure the vertices of $\partial(A_e)$ with basic color 1 are indicated by bold lines. The dash line in the figure shows the noose corresponding to $\partial(A_e)$. The vertices with basic color 1 are included in five disjoint component P_1, P_2, P_3, P_4 and P_5. Let we start assigning the black colors in clockwise order to these vertices starting from the vertex which is denoted by an arrow and is called *starting vertex*. For every component with more than one vertex in $\partial(A_e)$, the closest vertex (in clockwise order) to the starting vertex is assigned $1_{\lfloor}$ and the furthest vertex of component to the starting node is assigned $1_{\rfloor}$. The vertices of the component located between the vertices with colors $1_{\lfloor}$ and $1_{\rfloor}$ are assigned 1^*. For example Component

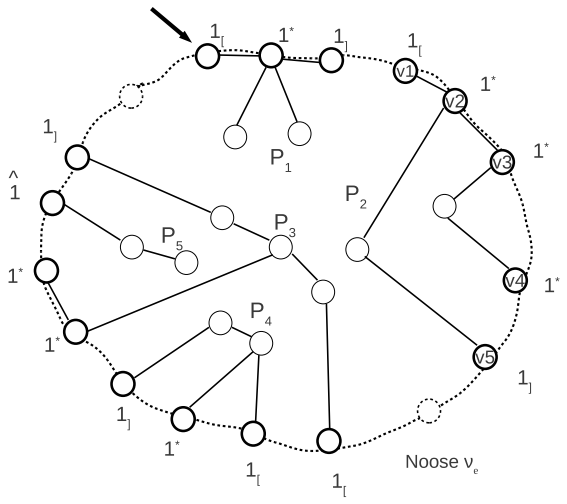


Figure 5.1: Assigning the proper black colors to $\partial(A_e)$

P_2 has five vertices in $\partial(A_e)$ with basic color 1, $v_1, v_2, ..., v_5$. Color $1_[$ is assigned to v_1 which is the closest vertex of P_2 to the starting vertex. Similarly $1_]$ is assigned to the vertex v_5. The middle vertices of P_2, which are located between v_1 and v_5, are colored by 1^*. If a connected component has only one vertex in $\partial(A_e)$ with basic color 1, the vertex is colored by $\hat{1}$. Component P_5 in Figure 5.1 is an example of a component with one black vertex in $\partial(A_e)$. Because H is a plane graph, $P_1, ..., P_r$ have the property of non-crossing partitions. From this, every partial solution of $H[A_e]$ is identified by a coloring of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$.

For a coloring $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$, we denote by $D_e(\eta)$ the partial solution identified by η with the minimum number of black vertices. In the merge step for the link $e = \{z, r\}$ incident to the root node r, we check the connectivity of $H[D_e(\eta)]$. A $D_e(\eta)$ with the minimum cardinality and $H[D_e(\eta)]$ connected is a minimum connected dominating set of H. For $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$, we define $a_e(\eta) = |D_e(\eta)|$ if η identifies a partial solution, otherwise $a_e(\eta) = +\infty$. For a leaf link e of T, $D_e(\eta)$ is computed for every $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$ by enumeration. For an internal link e of T, e has two child links e_1 and e_2. The sets $D_e(\eta)$ are computed by combining the sets of $D_{e_1}(\eta_1)$ and the sets of $D_{e_2}(\eta_2)$, where η_1 is a coloring of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$ and η_2 is a coloring of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_2}$.

Let $X_1 = \partial(A_e) \setminus \partial(A_{e_2})$, $X_2 = \partial(A_e) \setminus \partial(A_{e_1})$, $X_3 = \partial(A_e) \cap \partial(A_{e_1}) \cap \partial(A_{e_3})$, and $X_4 = (\partial(A_{e_1}) \cup \partial(A_{e_2})) \setminus \partial(A_e)$. Then $\partial(A_e) = X_1 \cup X_2 \cup X_3$, $\partial(A_{e_1}) = X_1 \cup X_3 \cup X_4$, and $\partial(A_{e_2}) = X_2 \cup X_3 \cup X_4$. A basic-coloring λ of $\partial(A_e)$ is formed from basic-colorings λ_1 of $\partial(A_{e_1})$ and basic colorings λ_2 of $\partial(A_{e_1})$ if:

1. For $u \in X_1$, $\lambda(u) = \lambda_1(u)$.
2. For $u \in X_2$, $\lambda(u) = \lambda_2(u)$.
3. For $u \in X_3$, if $\lambda_1(u) = \lambda_2(u) = 1$ then $\lambda(u) = 1$; if $\lambda_1(u) = \lambda_2(u) = \hat{0}$ then $\lambda(u) = \hat{0}$; and if $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$ then $\lambda(u) = 0$.
4. For $u \in X_4$, $\lambda_1(u) = \lambda_2(u) = 1$, or $\lambda_1(u) = 0$ and $\lambda_2(u) = \hat{0}$, or $\lambda_1(u) = \hat{0}$ and $\lambda_2(u) = 0$.

For a basic-coloring λ which is formed by two basic-colorings λ_1 and λ_2, we compute the disjoint components $P_1, ..., P_r$ of $H[D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)]$, where for $i = 1, 2$ $\eta_i(u) = \lambda_i(u)$ if $\lambda_i(u) \in \{0, \hat{0}\}$ and $\eta_i(u) \in \{1_[, 1_], 1^*, \hat{1}\}$ if $\lambda_i(u) = 1$. $D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)$ is called a *candidate* for $D_e(\eta)$ if each P_i has at least one vertex in $\partial(A_e)$. If $D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)$ is a candidate, we convert the color of u with $\lambda(u) = 1$ into one color of $\{1_[, 1_], 1^*, \hat{1}\}$ according to if u is the small end, the large end, a middle vertex, or the only vertex of $V(P_i) \cap \partial(A_e)$, respectively, to get a coloring $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$. Finally, $D_e(\eta)$ is a candidate $D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)$ with the minimum cardinality.

5.1.3 Index method

Now we describe an implementation of Step III (called the *index method*). Assume that the partial solutions in $H[A_{e_1}]$ and those in $H[A_{e_2}]$ have been found. We use a table B_1 with 6^{b_1} entries to keep the colorings of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$. The entries of B_1 are indexed by $1, 2, ..., 6^{b_1}$. We define a bijection from $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$ to $\{1, 2, ..., 6^{b_1}\}$ such that given a coloring $\eta_1 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$, the entry of B_1 for η_1 can be found in $O(1)$ time when 6^{b_1} can be expressed within $O(1)$ words of a computer. For a coloring $\eta_1 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$, we keep $a_{e_1}(\eta_1)$ and $D_{e_1}(\eta_1)$ in the entry for η_1. Similarly, we use a table B_2 with 6^{b_2} entries to keep the colorings $\eta_2 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_2}$.

From tables B_1 and B_2, we compute the partial solutions in $H[A_e]$. We create a table B with 6^b entries and set $a_e(\eta)$ to $+\infty$ for every entry. We first form the basic-colorings

of $\{0, \hat{0}, 1\}^b$ by checking B_1 and B_2. Recall that a vertex $u \in \partial(A_{e_1})$ has basic color 1 if $\eta_1(u) \in \{1_[, 1_], 1^*, \hat{1}\}$. We partition the entries of B_1 into 3^{b_1} groups, each group is identified by a basic-coloring of $\{0, \hat{0}, 1\}^{b_1}$ for the vertices in $\partial(A_{e_1})$. Similarly, we partition the entries of B_2 into 3^{b_2} groups. We say a group S_1 of B_1 and a group S_2 of B_2 *matching* if the basic-coloring of S_1 and the basic-coloring of S_2 form a basic-coloring $\lambda \in \{0, \hat{0}, 1\}^b$. Notice that $\partial(A_{e_1}) = X_1 \cup X_3 \cup X_4$ and $\partial(A_{e_2}) = X_2 \cup X_3 \cup X_4$. From this, given a group S_1 of B_1, the matching groups S_2's of B_2 can be decided by the basic colorings for S_1 on the vertices of $X_3 \cup X_4$.

For each coloring $\eta_1 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$ in a group S_1 of B_1, we choose every coloring $\eta_2 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_2}$ in every matching group S_2, compute the disjoint components of $H[D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)]$, find the coloring $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$ from η_1 and η_2, and calculate $a_e(\eta)$ as $a_e(\eta) = a_{e_1}(\eta_1) + a_{e_2}(\eta_2) - (X_3 \cup X_4)_{\#1}$, where $(X_3 \cup X_4)_{\#1}$ is the number of vertices in $X_3 \cup X_4$ which are colored by 1. For every η computed from η_1 and η_2 we keep the minimum $a_e(\eta)$ and the corresponding $D_e(\eta)$ which is computed from $D_{e_1}(\eta_1) \cup D_{e_2}(\eta_2)$. In what follows we analyze the running time of the DPBF Algorithm. In [43] these theorems are not proved and the explicit value of exponents are not defined.

Lemma 5.1.1 *[43] DPBF Algorithm solves the CDS problem for a plane graph G of n vertices in $O(2^{4.67\mathrm{bw}(G)}\gamma_c(G) + n^3)$ time and $O(6^{\mathrm{bw}(G)}\gamma_c(G))$ memory space.*

Proof: It takes $O(n^3)$ time in Step I to find a kernel H of G [72]. Because H is a subgraph of G, $\mathrm{bw}(H) \leq \mathrm{bw}(G)$. Step II takes $O((\gamma_c(G))^3)$ time to find a sphere-cut decomposition of H with width $\mathrm{bw}(H)$ since $|V(H)| = O(\gamma_c(G))$.

Notice that for plane graph H, $|X_3| \leq 2$. In the following analysis for the running time of Step III, we assume that $X_3 = \emptyset$. This assumption does not change the order of the running time. Let $l_1 = |X_1|, l_2 = |X_2|$, and $l_4 = |X_4|$. Let θ be a fixed coloring scheme which assigns a specific subset of m vertices of X_4 the basic-color 1. We say a coloring scheme η_1 for $X_1 \cup X_4$ is subject to θ if for each vertex $u \in X_4$, $\eta_1(u) = \theta(u)$ for $\theta(u) \in \{0, \hat{0}\}$, and $\eta_1(u) \in \{1_[, 1_], 1^*, \hat{1}\}$ for $\theta(u) = 1$. Let $Q(l_1, m)$ to be the number of coloring schemes for $X_1 \cup X_4$ subject to θ. Then $Q(l_1, m) = 6^{l_1}4^m$. Similarly, let $Q(l_2, m)$ be the number of coloring schemes for $X_2 \cup X_4$ subject to θ. Then $Q(l_2, m) = 6^{l_2}4^m$. A coloring scheme η_1 for $X_1 \cup X_4$ subject to θ and a coloring scheme η_2 for $X_2 \cup X_4$ subject to θ form a coloring scheme η for $X_1 \cup X_2$. Therefore, for a given basic color assignment θ for X_4 we need to check $Q(l_1, m)Q(l_2, m)$ pairs of the coloring schemes η_1 and η_2 for the merge step. From

this, the total number of coloring scheme pairs to be checked is

$$\begin{aligned} N &= \sum_{m=0}^{l_4} \left(\begin{array}{c} l_4 \\ m \end{array} \right) 2^{l_4-m} Q(l_1,m)Q(l_2,m) = \sum_{m=0}^{l_4} \left(\begin{array}{c} l_4 \\ m \end{array} \right) 2^{l_4-m} 6^{l_1} 4^m 6^{l_2} 4^m \\ &= 6^{l_1+l_2} 2^{l_4} \sum_{m=0}^{l_4} \left(\begin{array}{c} l_4 \\ m \end{array} \right) 2^{3m} = 6^{l_1+l_2} 2^{l_4} (1+8)^{l_4} = 2^{l_1+l_2+l_4} 3^{l_1+l_2+2l_4}. \end{aligned}$$

From $l_1 + l_2 \leq \text{bw}(H)$, $l_1 + l_4 \leq \text{bw}(H)$, $l_2 + l_4 \leq \text{bw}(H)$, we have $l_1 + l_2 + l_4 \leq 1.5\text{bw}(H)$ and $l_1 + l_2 + 2l_4 \leq 2\text{bw}(H)$. Therefore,

$$N \leq 2^{1.5\text{bw}(H)} 3^{2\text{bw}(H)} \leq 2^{4.67\text{bw}(H)}.$$

Since there are $O(|V(H)|)$ merge steps, $|V(H)| = O(\gamma_c(G))$ and $\text{bw}(H) \leq \text{bw}(G)$, Step III takes $O(2^{4.67\text{bw}(H)}|V(H)|) = O(2^{4.67\text{bw}(G)}\gamma_c(G))$ time.

Each of tables B_1, B_2, and B has size $O(6^{\text{bw}(H)}|V(H)|)$ and the memory space required for Algorithm is $O(6^{\text{bw}(G)}\gamma_c(G))$. []

The running time of DPBF Algorithm can be improved by a more complex analysis for Step III. For a given basic coloring θ with m specific vertices of X_4 assigned color 1, we say in the proof of Lemma 5.1.1 that

$$Q(l_1,m)Q(l_2,m) = 6^{l_1} 4^m 6^{l_2} 4^m = 6^{l_1+l_2} 16^m \tag{5.1}$$

pairs of colorings η_1 and η_2 need to be checked. However, not every pair of colorings η_1 and η_2 subject to θ can produce a partial solution of $H[A_e]$. For example, assume that $m = 1$ and $u \in X_4$ is the vertex with the basic color 1. If $\eta_1(u) = \eta_2(u) = \hat{1}$ then $D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)$ is not a candidate of $D_e(\eta)$ because the disjoint component of $H[D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)]$ that contains u does not have any vertex in $\partial(A_e)$. The components identified by $\eta_1(u) = \eta_2(u) = \hat{1}$ in the above example are called *forbidding components* [42, 43]. Excluding forbidding components can reduce the running time of Step III [42, 43]. It is difficult to identify all forbidding components. We work out some rules for identifying major forbidding components and show that the value of 16^m in Equation 5.1 can be improved to 14.8^m. By this improvement, we have the following result.

Theorem 5.1.1 *[43] DPBF Algorithm solves the CDS problem for a plane G of n vertices in $O(2^{4.62\text{bw}(G)}\gamma_c(G) + n^3)$ time.*

Proof: Let θ be a basic color scheme that assigns color 1 to m specific vertices of X_4. In what follows, using induction on m, we show that at most 14.8^m pairs of coloring (η_1, η_2) subject to θ can produce partial solutions. Some of the combinations of η_1 and η_2 create components that contain no vertex in X_1 and X_2. These components generate a partial dominating set that is not connected. Since the forbidding components consist of black vertices only, we use induction on m. Let the colorings η_1 and η_2 be ordered in clockwise and counter-clockwise order from the first vertex of X_4, respectively. For example in figure 5.2, c is the first vertex of X_4, the clockwise order of η_1 is $\{c, d, e, f, g, a, b\}$ and the counter-clockwise order of η_2 is $\{c, d, w, x, y, z\}$.

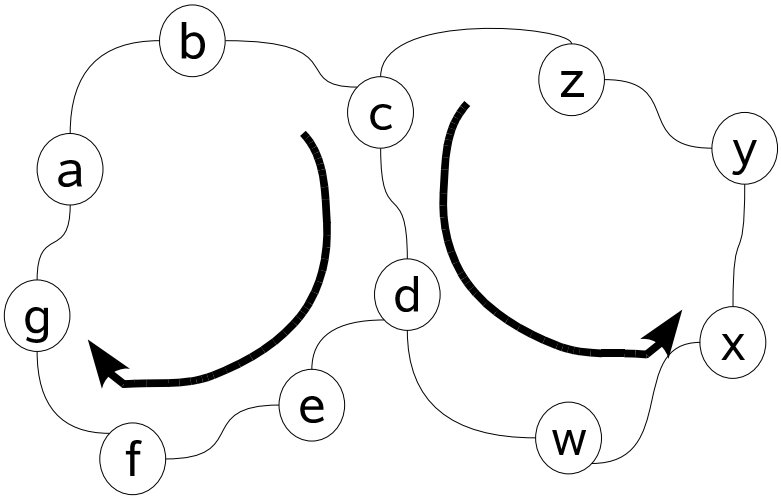


Figure 5.2: An example of clockwise and counter-clockwise order of vertices.

Assume that X_4 includes m vertices, $\{u_1, u_2, \ldots, u_m\}$ with basic color 1 in θ. For every vertex $u_i \in X_4$ the pair $(\eta_1(u_i), \eta_2(u_i))$ shows the color of u_i in the corresponding colorings η_1 and η_2 subject to θ. We define $B(i)$ as the set of black colorings for i vertices in X_4 that do not form a forbidding component. Every coloring in $B(i)$ is an i-tuple

$$((\eta_1(u_1), \eta_2(u_1)), (\eta_1(u_2), \eta_2(u_2), ..., (\eta_1(u_i), \eta_2(u_i)))$$

containing i pairs of colorings corresponding to i black vertices. Notice that $\eta_1(u_1), \eta_2(u_1) \in \{1_[, \hat{1}\}$ and $\eta_1(u_i), \eta_2(u_i) \in \{1_[, 1_], 1^*, \hat{1}\}$ for $2 \leq i \leq m$. Since $(\eta_1(u_i), \eta_2(u_i)) = (\hat{1}, \hat{1})$ creates a forbidding component, $B(1) = \{(1_[, 1_[), (1_[, \hat{1}), (\hat{1}, 1_[)\}$ and for every vertex u_i, $2 \leq i \leq m$

there are 15 possible pairs:

$$\{ \quad (1_[, 1_[), (1_[, 1^*), (1_[, 1_]), (1_[, \hat{1}), (1_], 1_[), (1_], 1^*)(1_], 1_]), (1_], \hat{1}),$$
$$(1^*, 1_[), (1^*, 1^*), (1^*, 1_]), (1^*, \hat{1}), (\hat{1}, 1_[), (\hat{1}, 1^*), (\hat{1}, 1_])\}.$$

Including these pairs to the existing colorings in $B(i)$ generates the colorings of $B(i+1)$. However, some of these pairs and some i-tuples of $B(i)$ may make an $(i+1)$-tuples results in a forbidding component. For instance, adding $(1_], \hat{1})$ to $(1_[, \hat{1})$ from $B(1)$ generates a forbidding component. For every coloring of $B(1)$, Table 5.1 shows the list of pairs of coloring that generate forbidding components.

Coloring in $B(1)$	Pairs resulting forbidding components
$(1_[, 1_[)$	$(1_], 1_])$
$(1_[, \hat{1})$	$(1_], \hat{1})$
$(\hat{1}, 1_[)$	$(\hat{1}, 1_])$

Table 5.1: The list of pairs that adding them to some colorings in $B(1)$ generates forbidding components

To generalize our method, we divide $B(i)$ into the following four classes:

1. $C_1(i)$ contains all the i-tuples, such that $(\eta_1(u_i), \eta_2(u_i)) = (1_[, 1_[)$.
2. $C_2(i)$ contains all the i-tuples, such that $(\eta_1(u_i), \eta_2(u_i)) = (1_[, \hat{1})$.
3. $C_3(i)$ contains all the i-tuples, such that $(\eta_1(u_i), \eta_2(u_i)) = (\hat{1}, 1_[)$.
4. $C_4(i)$ contains all other i-tuples.

We calculate $B(i+1)$ form $B(i)$ by adding all possible pairs of coloring for $(i+1)^{th}$ black vertex to $C_j(i)$ and generating $C_j(i+1)$, for $1 \leq j \leq 4$.

We generate $C_j(i+1)$ from $C_j(i)$ for $1 \leq j \leq 4$, as follows:

1. $C_1(i+1)$ is generated by adding $(1_[, 1_[)$ to $C_j(i)$, for $1 \leq j \leq 4$.
2. $C_2(i+1)$ is generated by adding $(1_[, \hat{1})$ to $C_j(i)$, for $1 \leq j \leq 4$.
3. $C_3(i+1)$ is generated by adding $(\hat{1}, 1_[)$ to $C_j(i)$, for $1 \leq j \leq 4$.
4. $C_4(i+1)$ is generated by adding :

(a) all remaining possible pairs except $(1_\rfloor, 1_\rfloor)$ to $C_1(i)$. (11 possible pairs)

(b) all remaining possible pairs except $(1_\rfloor, \hat{1})$ to $C_2(i)$. (11 possible pairs)

(c) all remaining possible pairs except $(\hat{1}, 1_\rfloor)$ to $C_3(i)$. (11 possible pairs)

(d) all remaining possible pairs to $C_4(i)$. (12 possible pairs)

Let $|C_j(i)|$ be the number of i-tuples in $C_j(i)$. Then we have

$$(|C_1(i+1)|, |C_2(i+1)|, |C_3(i+1)|, |C_4(i+1)|)^T \leq A(|C_1(i)|, |C_2(i)|, |C_3(i)|, |C_4(i)|)^T,$$

where

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 11 & 11 & 11 & 12 \end{pmatrix}$$

Let z be the largest real eigenvalue of A. Then

$$|B(i+1)| = \sum_{j=1}^{4} |C_j(i+1)| \leq z \sum_{j=1}^{4} |C_j(i)| = z|B(i)|.$$

From this, $z \leq 14.8$ and $|B(1)| = 3$, $|B(i+1)| \leq z^i |B(1)| = 3 \times (14.8)^i \leq (14.8)^{i+1}$.

When there are m vertices of X_4 are given the basic color 1, there are 16^m of pairs of colorings (η_1, η_2) for the m vertices but we need to check only the pairs in $B(m)$. So we can replace 16^m in Equation 5.1 by $(14.8)^m$. Therefore, the new value for N can be recalculated as follows.

$$N \leq \sum_{m=0}^{l_4} \binom{l_4}{m} 2^{l_4-m} 6^{l_1+l_2} (14.8)^m = 6^{l_1+l_2} (16.8)^{l_4}.$$

It is easy to see that N has the maximum value when $|\partial(A_e)| = |\partial(A_{e_1})| = |\partial(A_{e_2})|$. This can happen only when $l_1 = l_2 = l_4 = \mathrm{bw}(H)/2$. Therefore, $N \leq 2^{4.620\mathrm{bw}(H)}$ that gives $N \leq 2^{9.8\sqrt{n}}$.

[]

5.1.4 Fast matrix multiplication and distance product

In the index method, the partial solution $a_e(\eta)$ for each coloring η is calculated by checking every pairs of colorings η_1 and η_2 which form η and taking the minimum of $a_{e_1}(\eta_1) +$

$a_{e_2}(\eta_2) - (X_3 \cup X_4)_{\#1}$. The partial solutions can also be calculated by distance matrix multiplication [40]. To do so, we need to put each partial solution identified by a coloring $\eta_1 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$ to an element of a $p \times q$ matrix A and each partial solution identified by a coloring $\eta_2 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_2}$ to an element of a $q \times r$ matrix B such that each element $C[i,j] = \min_{k=1}^{q}\{A[i,k] + B[k,j]\}$ of $C = A \times B$ contains a partial solutions identified by a coloring $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b}$.

In contrast to the local problems like the DOMINATING SET problem, it is not straightforward to get the matrices A and B for the non-local problems like the CDS problem. One difficulty is as follows. For colorings η_1 and η_2 which form a coloring η, the colors of vertices in $X_1 \cup X_2 \cup X_3$ from η can not be decided only by the colors of vertices in $X_1 \cup X_3$ from η_1 and the colors of vertices in $X_2 \cup X_3$ from η_2. The information on the disjoint connected components in the partial solution of $H[D_{e_1}(\eta_1) \cup D_{e_2}(\eta_2)]$ is further required to decide η. The computation of the disjoint connected components is known as the *post-processing* for the non-local problems in [40].

One approach to overcome this difficulty is to get the information of disjoint connected components by a *pre-processing*: for each coloring η, we identify all pairs of η_1 and η_2 which form η and put the partial solutions of η_1 to a specific row of A and the partial solutions of η_2 to a specific cloumn of B. More specifically (for simplicity we assume that $X_3 = \emptyset$), let A be the matrix of $p = 6^{|X_1|}$ rows for the partial solutions by η_1 and B be the matrix of $r = 6^{|X_2|}$ columns for the partial solutions by η_2. Each row of A is indexed by a color of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{|X_1|}$ and each column of B is indexed by a color of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{|X_2|}$. For a coloring η, assume that the ith row of A corresponds to the colors of vertices in X_1 from η and the jth column corresponds to the colors of vertices in X_2 from η. Assume that there are q pairs of η_1 and η_2 which form η. For each pair of η_1 and η_2, we put the partial solution by η_1 $(a_{e_1}(\eta_1) - (X_4)_{\#1})$ to an element in the ith row of A and the partial solution by η_2 (a_{e_2}) to an element in the jth column of B such that $(A[i,k], B[k,j])$, $1 \leq k \leq q$, correspond to the q pairs of η_1 and η_2. Then $C[i,j] = \min_{k=1}^{q}\{A[i,k] + B[k,j]\}$ gives the partial solution by η. It can be shown that $q \leq (14.8)^{|X_4|}$ and the running time for the pre-processing is $O((14.8)^{|X_4|})$. The details of the pre-processing is given in Appendix B.

For $X_3 \neq \emptyset$, we compute $C = A \times B$ for each coloring of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{|X_3|}$. Since $|X_3| \leq 2$, there are at most 6^2 such computations.

If the conventional distance matrix multiplication is used, it takes $O(p \times q \times r)$ time to compute the distance product $C = A \times B$. Since A and B have integer values, the distance

product can be calculated by integer matrix multiplication. Let $O(n^\omega)$ be the time for calculating the integer product of two $n \times n$ matrices. It is known that distance product of $C = A \times B$ can be calculated in $O(s(pr)^{(\omega-1)/2}q)$ time if the values of A and B are between $-s$ and s [111]. The best known upper bound on ω is 2.376. From this, the running time of DPBF Algorithm is improved to $O(2^{3.722\mathrm{bw}(G)}\gamma_c(G) + n^3)$ and $O(2^{8.08\sqrt{n}}\gamma_c(G) + n^3)$. This result is summarized in the following theorem.

Theorem 5.1.2 *DPBF Algorithm solves the CDS problem for a plane graph G of n vertices in $O(2^{3.722\mathrm{bw}(G)}\gamma_c(G) + n^3)$ and $O(2^{8.08\sqrt{n}}\gamma_c(G) + n^3)$ time.*

Notice that the improvement by fast matrix multiplication is only of theoretical interests. The method is not practical [87].

5.2 Fixed parameter time complexity and bidimensionality

As we mentioned in previous section the running time of DPBF algorithm grows exponentially in $bw(G)$. A straightforward upperbound $bw(G) \leq c\sqrt{\gamma(G)} \leq c\sqrt{\gamma_c(G)}$, gives running time $O(2^{23.7\sqrt{\gamma_c(G)}}\gamma_c(G) + n^3)$. We prove a better upperbound on $bw(G)$, $\mathrm{bw}(G) \leq 2\sqrt{10\gamma_c(G)} + 32$.
From this and Theorem 5.1.1, the CDS problem admits an $O(2^{23.54\sqrt{\gamma_c(G)}}\gamma_c(G) + n^3)$ time fixed-parameter algorithm. We also introduce a new value for the density of the PLANAR CDS problem in bidimensionality theorem.

5.2.1 Fixed parameter time complexity

A $k \times h$ cylinder $C_{k,h}$ is a Cartesian product of a cycle of k vertices and a path of h vertices. Its vertex set is $V(C_{k,h}) = \{(i,j)|0 \leq i < h, 0 \leq j < k\}$ and there is an edge between vertices (i,j) and (i',j') if and only if $i = i'$ and $|j - j'| \equiv 1(\mathrm{mod}k)$ or $j = j'$ and $(i - i') = \pm 1$. We denote by C_i the cycle induced by the vertex set $\{(i,j)|0 \leq j < k\}$. Figure 5.3 shows a 18×10 cylinder.

Based on the following lemma which is proved in [62], we prove a better upperbound on $bw(G)$.

Lemma 5.2.1 *[62] Let G be a planar graph and k, h be integers with $k \geq 3$ and $h \geq 1$. Then G has either a minor isomorphic to a $k \times h$ cylinder or $\mathrm{bw}(G) \leq k + 2h - 3$.*

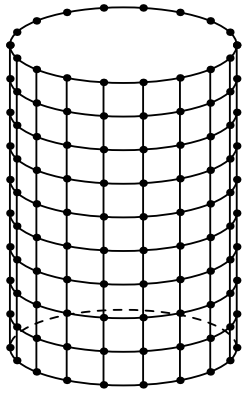


Figure 5.3: 18×10 cylinder $C_{18,10}$

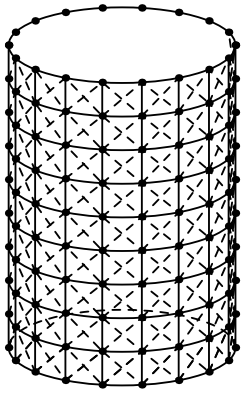


Figure 5.4: Supplement graph of $C_{18,10}$

We define the *supplement graph* of $C_{k,h}$ to be the graph $G_{k,h}$ with $V(G_{k,h}) = V(C_{k,h})$ and

$$\begin{aligned} E(G_{k,h}) &= E(C_{k,h})) \cup \\ &\{\{(i,j),(i+1,j+1(\mathrm{mod}k))\},\{(i+1,j),(i,j+1(\mathrm{mod}k)\}|0 \le i < h-1, 0 \le j < k\}. \end{aligned}$$

Figure 5.4 shows the supplement graph of $C_{18,10}$.

Lemma 5.2.2 *For a plane graph G which has a minor isomorphic to $C_{k,h}$ $(k \ge 2, h \ge 10)$, let H be the subgraph of the supplement graph $G_{k,h}$ induced by the vertex set $\{(i,j)|4 \le i \le h-5\}$. Then $\gamma_c(H) \le \gamma_c(G)$ (See Figure 5.5).*

Proof: To show the lemma, we prove that given a minimum CDS D of G, we can construct a CDS D' of H with $|D'| \le |D|$.

We do not distinguish $C_{k,h}$ from the minor of G isomorphic to $C_{k,h}$. We also view $C_{k,h}$ a plane graph with an embedding obtained naturally from the embedding of G. Let f_t be the top face of $C_{k,h}$ (the face incident only to the vertices of C_0) and f_b the bottom face of $C_{k,h}$ (the face incident only to the vertices of C_{h-1}). In Figure 5.5 top and bottom faces are indicated by grey color.

Let $W(f_t) = \{u|u \in V(G)$ and u is in $f_t\}$ and let $W(f_b) = \{u|u \in V(G)$ and u is in $f_b\}$. Let $D_1 = D \setminus (W(f_t) \cup W(f_b))$. We define $\phi : D_1 \to V(G_{k,h})$ as follows.

- If $u \in D_1 \cap V(G_{k,h})$ then $\phi(u) = u$.
- If $u \in D_1 \setminus V(G_{k,h})$ and u is in a face $f \in F(C_{k,h}) \setminus \{f_t, f_b\}$ then $\phi(u)$ is a vertex v of $V(f)$.

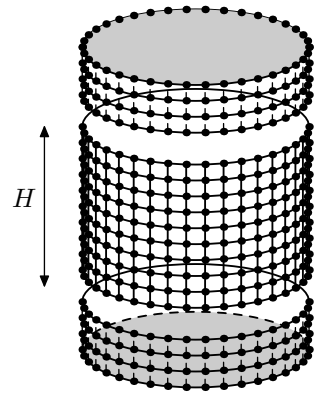


Figure 5.5: Graph H is a subgraph of $G_{20,18}$ and corresponding top and bottom faces are colored by grey.

- If $u \in D_1 \setminus V(G_{k,h})$ and u is incident to $f_1, f_2 \in F(G_{k,h})$ then $\phi(u)$ is a vertex v of $V(f_1) \cap V(f_2)$.

Let $D_2 = \{\phi(u) | u \in D_1\}$. Then the followings hold.

1. $|D_2| \leq |D_1|$.
2. $V(G_{k,h}) \cap N[D_1] \subseteq V(G_{k,h}) \cap N[D_2]$.
3. For any $D_1' \subseteq D_1$ and $D_2' = \{\phi(u) | u \in D_1'\}$, if $G[D_1']$ is connected then $G_{k,h}[D_2']$ is connected.

The rest of the proof is partitioned into two cases.

Case 1: $D \subseteq V(G_{k,h})$. From (2) and (3), D_2 is a CDS of $G_{k,h}$. For each vertex (i, j) ($i = 0, 1, 2$ or 3) of D_2 on cycle C_i, we replace (i, j) with vertex $(4, j)$. For each vertex (i, j) ($i = h-1, h-2, h-3$ or $h-4$) of D_2 on cycle C_i, we replace (i, j) with vertex $(h-5, j)$. Then we get a CDS D' of H such that $D' \subseteq V(H)$ and $|D'| = |D_2| \leq |D_1| \leq |D|$.

Case 2: $D \not\subseteq V(G_{k,h})$. Since no vertex (i, j) where $1 \leq i \leq h-2$ cannot be dominated by any vertex of $W(f_t) \cup W(f_b)$, D_1 dominates all vertices $(i, j), 1 \leq i \leq h-2, 0 \leq j < k$. Based on this and (2), D_2 dominates all vertices $(i, j), 1 \leq i \leq h-2, 0 \leq j < k$. On the other hand, the subgraph of $G_{k,h}$ induced by D_2 may not be connected. Each component of $G_{k,h}[D_2]$ may be further partitioned into multiple sub-components in graph H. We show that we can construct a CDS D' of H by adding some vertices to connect all the sub-components and also the number of added vertices is at most $|D_2 \setminus V(H)|$.

For a connected component X of $G_{k,h}[D_2]$, since $G[D]$ is connected and given (3), we have $|V(X)\cap\{(0,j)|0\leq j<k\}|+|V(X)\cap\{(h-1,j)|0\leq j<k\}|\geq 1$. A component is a *top-component* if $|V(X)\cap\{(0,j)|0\leq j<k\}|\geq 1$, otherwise it is a *bottom-component*. For every top-component X define $u_x\in V(X)\cap\{(0,j)|0\leq j<k\}$ and for every bottom-component X, u_x is defined as $u_x\in V(X)\cap\{(h-1,j)|0\leq j<k\}$.

For every top-component X we define

$$\begin{aligned} W_X &= \{(i,j)|(i,j)\in V(X)\cap V(H)\}\cup\{(4,j)|(i,j)\in V(X), i=3\}\cup \\ &\quad \{(h-5,j)|(i,j)\in V(X), i=h-4,h-3,h-2,h-1\}. \\ T_X &= \{(i,j)|(i,j)\in V(X)0\leq i\leq 2\} \end{aligned}$$

and similarly for every bottom-component X, W_X and B_X are defined as

$$\begin{aligned} W_X &= \{(i,j)|(i,j)\in V(X)\cap V(H)\}\cup\{(h-5,j)|(i,j)\in V(X), i=h-4\}\cup \\ &\quad \{(4,j)|(i,j)\in V(X)i=1,2,3\}. \\ B_X &= \{(i,j)|(i,j)\in V(X)h-3\leq i\leq h-1\} \end{aligned}$$

Let $W=\bigcup W_X$ for all components X of $G_{k,h}[D_2]$. We define $G_T=\bigcup T_X$ for all top-components of $G_{k,h}[D_2]$, and similarly G_B is defined as $G_B=\bigcup B_X$ for all bottom-components of $G_{k,h}[D_2]$. Then $W\subseteq V(H)$ is a dominating set of H and $|W|\leq|(D_2\cap V(G_{k,h}))\setminus(V(G_T)\cup V(G_B))|$. If $H[W]$ is connected then the lemma holds. Otherwise, we show that a CDS D' of H can be constructed by adding at most $|(V(G_T)\cup V(G_B))|$ vertices to connect the components of $H[W]$. Since G_T and G_B are two disjoint graphs, $|(V(G_T)\cup V(G_B))|=|V(G_T)|+|V(G_B)|$. We show that the minimum sizes of $|V(G_T)|$ and $|V(G_B)|$ provide enough vertices to connect components of $H[W]$. In what follows we compute a lower bound for $|V(G_T)|$ using the top-components. A lower bound for $|V(G_B)|$ can be found similarly using the bottom-components.

Let X be a connected top-component of $G_{k,h}[D_2]$ with r sub-components $Y_1,...,Y_r$ in $H[W]$. For each sub-component Y_a there is a vertex $y_a=(4,j)$ of Y_a such that $x_a=(3,j)$ is a vertex of X and each vertex $x_a(1\leq a\leq r)$ is incident to a vertex of tree T_X of X in G_T. T_X is a tree that connects $Y_1,...,Y_r$ together and connect them to u_x.

We define the *debit* of a sub-component $Y_a(2\leq a\leq r)$, indicated by $debit(Y_a)$, as the number of columns between x_a and x_{a-1}. Based on the debit of each sub-component we define the following two types of sub-components:

- close sub-component: A sub-component Y_α whose $debit(Y_\alpha) = 1$ for $2 \leq \alpha \leq r$.
- far sub-component: A sub-component Y_α whose $debit(Y_\alpha) > 1$ for $2 \leq \alpha \leq r$.

To connect $Y_1, ..., Y_r$, we need at least $debit(Y_2) + ... + debit(Y_r)$ vertices in T_X. The following lower bound can be defined for $|V(T_X)|$ which is known as the credit that T_X provides for X to connect its sub-components and indicated by $credit(T_X)$.

$$credit(T_X) = |V(T_X)| \geq debit(Y_2) + ... + debit(Y_r).$$

If a sub-component Y_a is a close sub-component, it means that there is only one column, called *middle column* between y_a and y_{a-1}. T_X clearly has a vertex on the middle column. Using this vertex and adding a new vertex in W between Y_a and Y_{a-1}, we connect every close sub-component Y_a to sub-component Y_{a-1}. In formal, for every component X and for every close sub-component $Y_a \in X$ with middle column l_m we define D' as follows:

$$D' = W \cup \{(4, l_m) | (i, l_m) \in V(T_X)\} \tag{5.2}$$

Once connecting the close sub-components, for every component X containing at least one close sub-component, we get a new component X' such that X' has only r' far sub-components in H. In what follows we show that $credit(T_{X'})$ is at least $2r'$ and enough to make D' connected.

Let X' be the result of connecting c close sub-components in X. Let X' has r' new sub-components $Y'_1, ..., Y'_{r'}$ which are all far sub-components. $credit(T_{X'})$ is defined as

$$credit(T_{X'}) = (credit(T_X) - c) = (|V(T_X)| - c) \geq debit(Y'_2) + ... + debit(Y'_{r'}) \geq 2(r' - 1) \tag{5.3}$$

In the remaining of this proof we improve the lower bound to $credit(T_{X'}) \geq 2r'$. We consider three different cases for X'.

1. X' contains at least one sub-component Y'_a with $debit(Y'_a) \geq 4$. In this case it is obvious from 5.3 that $credit(T_{X'}) \geq 2r'$.
2. X' contains at least one sub-component Y'_a with $debit(Y'_a) = 3$. Since in this case we assumed that there are at most three columns between every two sub-components in X' there must be at least one column with more than one vertex (including u'_x) in $T_{X'}$. Otherwise the sub-components can not be connected to $u_{x'}$. Based on this and 5.3, $credit(T_{X'}) \geq 2r'$.

3. X' contains only sub-components Y'_a with $debit(Y'_a) = 2$. Similar to the second case, if $T_{X'}$ has only one vertex on each column, only the connection between sub-components is possible and to connect them to $u_{x'}$, two more vertices are needed. Based on this and 5.3, $credit(T_{X'}) \geq 2r'$.

This lower bound also holds for a component X' with only one sub-component ($r' = 1$), since x'_1 is connected to $u_{x'}$ by a path in $T_{X'}$ with three vertices, $credit(T_{X'}) \geq 2$.

Therefor, for each component X' with r' far sub-components, we have

$$credit(T_{X'}) \geq 2r'. \tag{5.4}$$

Let $G_{k,h}[D_2]$ contains m connected components $X_1, ..., X_m$, and assume after connecting close sub-components to their neighbours, each resulting X'_i $1 \leq i \leq m$ has l_i far sub-components. Since $H[W]$ is a dominating set of H, there are at most two columns distance between sub-components. Therefore, the number of far sub-components in $H[W]$ is $n = l_1 + \ldots + l_m$. Thus, we need at most $2n - 2$ vertices to connect these sub-components and make $H[W]$ a connected dominating set of H. Based on lower bound in 5.4 we have

$$credit(T_{X'_1}) + \ldots + credit(T_{X'_m}) \geq 2l_1 + \ldots + 2l_m \geq 2n$$

Therefore, the total credits of components is at least $2n$.

Summarizing the above, each component X in $G_{k,h}[D_2]$ containing b sub-components in $H[W]$, contributes $2b$ credits. Therefore, a CDS D' of H can be constructed by adding at most $|(V(G_T) \cup V(G_B))|$ vertices to connect the components of $H[W]$. This completes the proof of the lemma.

[]

Lemma 5.2.3 $\gamma_c(G_{k,h}) \geq (k \times h - 4)/5$.

Proof: Let D be a CDS of $G_{k,h}$. Then $G_{k,h}[D]$ contains a tree T which connects all vertices of D. We choose a non-leaf vertex of T as the root. Then the root can dominate at most 9 vertices including itself. We scan T in the level-order. Every vertex u during the scan can dominate at most five vertices which have not been dominated by the scanned vertices. So $|D| \geq 1 + (k \times h - 9)/5 = (k \times h - 4)/5$. []

Theorem 5.2.1 *For a plane graph G,* $\mathrm{bw}(G) \leq 2\sqrt{10\gamma_c(G)} + 32$.

Proof: Assume for contradiction that $\mathrm{bw}(G) > 2\sqrt{10\gamma_c(G)} + 32$. By Lemma 5.2.1, G has a cyliner minor $C_{k,k/2}$ with $k > \sqrt{10\gamma_c(G)} + 17$. By Lemmas 5.2.2 and 5.2.3,

$$\gamma_c(G) \geq \gamma_c(G_{k,\frac{k}{2}-8}) \geq \frac{k(\frac{k}{2}-8)-4}{5}.$$

Base on this, $k < \sqrt{10\gamma_c(G)} + 17$, a contradiction. []

5.2.2 Bidimensionality

The main tool used in the design of most subexponential fixed-parameter algorithms for minor-closed graph classes such as planar graphs is based on the following fact. For every graph G in minor-closed graph class if the size of the optimal solution is at most k, then tree/branchwidth is bounded by some function of k called $f(k)$. In many cases $f(k)$ is sublinear in k, often $O(\sqrt{k})$. For example in the previous section we use the same tool to prove the time complexity of DPBF algorithm for PLANAR CDS problem.

Bidimensionality, introduced in series of papers [38, 34, 33, 35], provides a tool for developing subexponential fixed-parameter algorithms for optimization problems on graph families that exclude a minor. A minor of a graph G is any graph H that is isomorphic to a graph that can be obtained from a subgraph of G by contracting some edges. If G does not have a graph H as a minor, then G is called H-minor free. There are two types of bidimensionality: *minor-bidimensionality, contraction-bidimensionality.* For any problem in one of these two bidimensionalities, their corresponding algorithms on planar graphs can be extended to some larger families of graphs. In what follows we give a formal definition of bidimensionality, and we show that our results in previous section can improve some parameters in bidimensionality theorem for the CDS problem.

A *graph parameter* P is a function mapping graphs to non-negative integers, such as size of dominating set. The *parametrized problem associated with* P, for a fixed k asks whether $P(G) \leq k$ for a given graph G. A *partially triangulated* $(r \times r)$*-grid* is any planar graph obtained by adding edges between pairs of nonconsecutive vertices on a common face of a planar embedding of an $(r \times r)$-grid.

Definition: [35] A parameter P is minor-bidimensional with density δ if

1. contracting or deleting an edge in a graph G cannot increase $P(G)$, and

2. for the $(r \times r)$-grid R, $P(R) = (\delta r)^2 + o((\delta r)^2)$.

In [35], it is proved that if an optimization problem is minor- bidimensional, there is a subexponential fixed-parameter algorithm for the problem on H-minor-free graphs for any fixed graph H.

Definition: [35]A parameter P is contraction-bidimensional with density δ if

1. contracting an edge in a graph G cannot increase $P(G)$, and
2. for any partially triangulated $(r \times r)$-grid R, $P(R) \geq (\delta r)^2 + o((\delta r)^2)$, and
3. δ is the smallest real number for which inequality holds.

If an optimization problem is contraction-bidimensional, there is a subexponential fixed parameter algorithm for the problem on graphs from apex-minor-free family. Apex-minor-free means H-minor-free where H is a graph in which the removal of one vertex makes the graph planar.

The CDS problem is contraction-bidimensional [35]. A known density of this problem is $\frac{1}{3}$ which is same as the density of DOMINATING SET problem. This density provides an lower bound for the density of CDS problem:

Theorem 5.2.2 *The CDS problem is contraction-bidimensional problem with density δ at least $\frac{1}{\sqrt{5}}$.*

Proof:

Clearly, edge contraction cannot increase the size of CDS. Let R be a partially triangulated $(r \times r)$-grid. The proof for the second condition, $P(R) \geq \frac{r^2}{5} + o(\frac{r^2}{5})$, is similar to the proof of Lemma 5.2.3. []

In addition, $\frac{1}{\sqrt{5}}$ is the smallest value for δ since for any partially triangulated $(r \times r)$-grid R, there is a dominating set of size $\frac{r^2}{5} + O(r)$. Appendix C describes how this dominating set can be constructed. Combine this and Theorem 5.2.2 we have following theorem:

Theorem 5.2.3 *The CDS problem is contraction-bidimensional problem with density $\delta = \frac{1}{\sqrt{5}}$.*

5.3 Computational results

We tested the performance of DPBF Algorithm on six classes of planar graphs. These graphs have been used in the previous computational studies for the PLANAR DOMINATING SET problem [3, 4, 87]. Class (1) is a set of random maximal planar graphs and their subgraphs generated by LEDA [1]. Class (2) includes the Delaunay triangulations of point sets taken from TSPLIB [95]. The instances of Classes (3) and (4) are the triangulations and intersection graphs generated by LEDA, respectively. The instances of Class (5) are Gabriel graphs generated using the points uniformly distributed in a two-dimensional plane. The instances of Class (6) are random planar graphs generated by the PIGALE library [2].

We use the reduction rules of [72] to compute the kernels of input instances in Step I and the $O(n^3)$ time algorithm of [17] to compute optimal sphere-cut decompositions of kernels in Step II. For Step III, we use an indexing method to access the tables. To save memory, we compute the colorings η and sets $D_e(\eta)$ for each link e of T in the post-order manner. Once the colorings η and sets $D_e(\eta)$ are computed for a link e, the solutions for the child links of e are discarded. We have tested the implementations without and with excluding forbidding components. The results show that the simple version (without considering forbidding components) has a better performance. Because the fast distance matrix multiplication is not practical [87], applying this technique does not improve the practical performance of the algorithm.

The computer used for testing 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++.

Table 5.2 shows the computational results of the simple version of DPBF Algorithm. H is the kernel of an instance computed in Step I. In Step II, an optimal sphere cut decomposition of H is computed and we report $|E(H)|$, the size of H, the branchwidth $\mathrm{bw}(H)$ of H, and the running time of this step. For Step III, we give $\gamma_c(G)$ obtained, the running time of the step and the required memory in mega bytes (MB). All times in the table are in seconds.

Now we go over the details of our results. It is shown in Appendix A that the branchwidth of the instances of class (1) is at most four. Our results show that reduction rules are very effective on these graphs and that the size of the kernels is much smaller than the size of the original graphs. Thus, step III is fast and the minimum CDS of some instances with 16000 edges can be computed in about one hour on our platform.

However, the branchwidth increases very fast in the size of the graph for the instances of Classes (2) and (5). In addition, the reduction rules do not reduce the size of the original graphs very much, and the size and branchwidth of generated kernels are the same as those of the original graphs. The running time of Step III increases significantly with the branchwidth of instances (e.g., see the running time of instances pr144 and kroB150). For instances with the same branchwidth the running time of this step depends on the size of the kernel (see instances eil51 and lin105). For these classes of planar graphs DPBF Algorithm is time consuming and can solve the CDS problem on instances of size up to a few hundreds edges in a practical time.

The branchwidth of instances of Classes (3) and (4) grows relatively slow in the instance sizes. Furthermore, data reduction rules are effective on the instances of Class (4). The branchwidth of graph instances in Class (6) does not grow in the instance size thus, DPBF Algorithm is efficient for this class.

The memory space required by DPBF Algorithm in step III is a bottleneck for solving instances with large branchwidth. Experimental results show that DPBF Algorithm can compute a minimum CDS for instances with the branchwidth of kernels at most 10 ($\text{bw}(H) \leq 10$) using 3GBytes of memory space.

Our computational results confirm the theoretical analysis of DPBF Algorithm. It is efficient for graphs with small branchwidth but time and memory consuming for graphs with large branchwidth. This suggests that the branchwidth of a planar graph is a key parameter to decide if a problem on the graph can be solved efficiently or not. For example, Class (1) graphs have branchwidth at most four and thus admit efficient algorithms for many hard problems. On the other hand, the problems on graphs in Classes (2) and (5) are less tractable due to the large branchwidth of the instances.

Class	Graph G	$\|E(G)\|$	$bw(G)$	Step I	Step II			Step III		total	maximum
				time	$\|E(H)\|$	bw(H)	time	$\gamma_c(G)$	time	time	memory
(1)	max1000	2912	4	19.4	704	4	2.3	131	4	25.7	
	max2000	5978	4	63	1133	4	6.0	252	9.9	78.9	
	max3000	8510	4	359	2531	4	37.6	417	94	491	
	max4000	10759	4	836	3965	4	145	614	458	1439	
	max5000	14311	4	848	3873	4	160	650	383	1392	
	max5000	16206	4	1702	5989	4	325	907	1769	3796	
(2)	eil51	140	8	0.1	140	8	0.2	14	253	254	0.03
	lin105	292	8	0.3	275	8	3	27	810	813	0.03
	pr144	393	9	1	347	7	0.5	25	18.1	19.7	0.06
	kroB150	436	10	1	436	10	0.8	36	133856	133858	1.05
	pr226	586	7	1.3	399	6	1.7	24	5.1	8.1	0.04
	ch130	377	10	0.3	377	10	0.6	34	38562	38563	0.74
(3)	tri100	288	7	0.7	258	6	0.6	20	7.1	8.4	0.05
	tri500	1470	7	10.1	1438	6	37.2	91	62.6	110	0.07
	tri800	2374	8	18	2279	7	86.4	149	289	393	0.13
	tri2000	5977	8	109	5751	8	603	369	5643	6355	0.48
	tri4000	11969	9	547	11236	9	3690	753	42323	46560	0.57
(4)	rand100	121	5	0.1	73	3	0.1	40	0.1	0.3	0.03
	rand500	709	7	1.7	545	6	0.4	216	10.8	12.9	0.05
	rand700	1037	7	2.9	836	6	1	301	17.8	21.8	0.07
	rand1000	1512	8	4.5	1242	7	2.5	421	422.8	429.8	0.25
	rand2000	3247	8	17.5	2852	8	17.8	839	10179	10214	0.38
	rand3000*	4943	10	-	-	10	-	-	-	-	
(5)	Gab50	88	4	0.1	88	4	0.1	22	0.2	0.4	0.03
	Gab100	182	7	0.1	179	7	0.3	41	66.7	67.1	0.11
	Gab200	366	8	0.7	362	8	1.5	81	2290	2293	0.13
	Gab300	552	10	1.4	545	10	1.6	121	12 days	12 days	2.53
(6)	P206	269	4	0.6	163	4	0.3	78	0.3	1.2	0.02
	P495	852	5	3.2	765	5	8.4	167	11.9	23.5	0.02
	P855	1434	6	7.9	1280	6	15.1	289	77.9	101	0.06
	P1000	1325	5	4.4	777	5	2.5	378	7.3	14.2	0.07
	P2000	2619	6	24.5	1527	6	12.3	738	58.0	94.8	0.11
	P4206	7101	6	256	6377	6	1816	1423	2411	4482	0.43

Table 5.2: Computational results (time in seconds) of DPBF Algorithm. For the instances marked with "*", the 3GByte memory is not enough for computing a minimum connected dominating set.

Chapter 6

Polynomial Time Approximation Schemes(PTAS) for planar graphs problems

6.1 Related methods

Based on the assumption $P \neq NP$ there is no efficient (polynomial time) algorithm for a NP-hard problem. If an almost optimal solution is good enough, approximation techniques are of the fastest methods. Readers may refer to [108] for a survey on approximation algorithms. Approximation ratio of an algorithm is the best measure to evaluate the algorithm. An extensive amount of studies have been devoted to improve the approximation ratio of algorithms for NP-hard problems, however in many cases, improving approximation ratio is not possible unless some collapses occur between complexity classes. For example Arora et al. in [11] prove that a class of NP-hard problems including vertex cover, maximum satisfiability, maximum cut, metric TSP, Steiner trees and shortest superstring does not have a PTAS, unless $P = NP$. Studies on NP-complete problems show that there are many problems which are easier to approximate on planar graphs. As an example maximum independent set is inapproximable within a factor of $n^{1/2\epsilon}$ for any $\epsilon > 0$ unless $P = NP$, while for planar graphs there is a 4-approximation algorithm. Another example is the DOMINATING SET problem defined on a general graph with n nodes. The DOMINATING SET problem is known to be $(1 + \log n)$-approximable [74], but not approximable within

a factor of $(1-\epsilon)\ln n$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\log\log n})$ [56]. The problem is also known to be fixed-parameter intractable unless the parametrized complexity classes collapse [44, 45], however the PLANAR DOMINATING SET problem has a PTAS.

There are two main methods for finding PTASs for NP-hard problems on planar graphs: *separator method* and *outer-planar decomposition method*. In the following sections we introduce these two methods and show how the maximum independent set problem in planar graphs can be solved using these methods. We also explain how branch-decomposition based algorithms for the PLANAR DOMINATING SET problem can be applied in the outer-planar decomposition method to generate a PTAS for the PLANAR DOMINATING SET problem.

6.1.1 Planar separator method

This method is based on the following theorem introduced by Lipton and Tarjan [81].

Planar separator theorem [81]: For any planar graph with n nodes there is a separator of size $O(\sqrt{n})$, whose removal splits the graph into subgraphs of size at most $2/3n$. Finding this separator can be done in polynomial time. Figure 6.2 shows an example of a planar separator of size 5 for a planar graph with 20 nodes.

The separator method recursively applies this theorem until the the size of resulting subgraphs is a constant such as $1/\epsilon$. The problem on these subgraphs is solved by brute force , then these partial solutions are combined through the recursion tree of separators.

Planar maximum independent set problem: Based on this method, Lipton and Tarjan in [82] gave an $1-O(1/\sqrt{\log n})$ approximation algorithm for the planar independent set problem with running time $O(n\log n)$. The algorithm works as follows. Let G be a planar graph with n vertices and $\epsilon = (\log\log n)/n$. For every vertex we define a cost as $1/n$. The algorithm has two steps:

1. Recursively applies the planar separator theorem, until the graph G' resulted from removing separators has no connected component with more than $\log\log n$ vertices. It is proved that the total size of separators in this step is $O(n/\sqrt{\log\log n})$.

2. In each connected component of G', finds a maximum independent set by checking every subset of vertices for independence. It then forms I as a union of maximum independent sets.

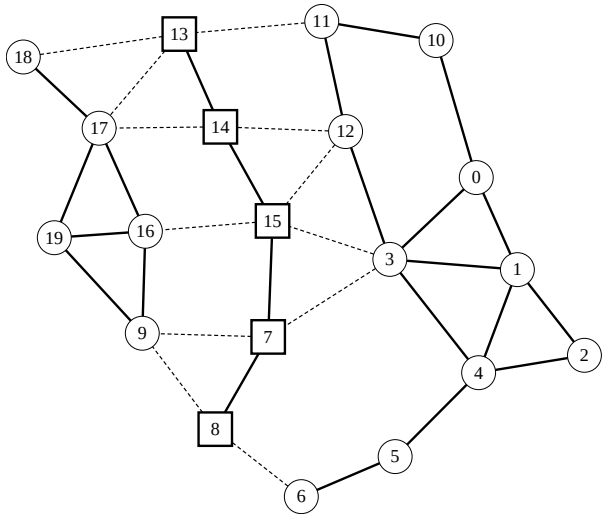


Figure 6.1: A separator of size of 5 for a planar graph with 20 nodes

Using the 4-colour theorem, Lipton and Tarjan show that the relative error of this algorithm is $O(1/\sqrt{\log \log n})$. The running time of the first step is $O(n \log n)$. Step 2 requires $O(n_i 2^{n_i})$ time on a connected component of n_i vertices. The total time required by Step 2 is thus $n \max\{\log n, 2^{\log \log n}\} = O(n \log n)$.

The error induced by this method is bounded by the total size of all separators, which is bounded by n. This approach can generate a PTAS for the problem if the optimal solution is at least some constant factor times n.

This method has two disadvantages :

1. This method gives a PTAS only if the optimal solution is a constant factor times n. However, this property does not hold for all NP-hard problems on planar graphs. For example, Grohe [60] states that the PLANAR DOMINATING SET problem is a problem to which the technique based on the separator theorem does not apply. In addition, for some problems such as independent set and vertex cover this upper bound for the optimal solution can only be achieved through the application of reduction rules or pruning methods.

2. This method is not practical. For example Lipton and Tarjan applied this method to obtain an approximation algorithm for maximum independent set problem.

   The approximation algorithm that Lipton and Tarjan developed for solving the maximum independent set problem, requires $O(n \max\{\log n, 2^{f(n)}\})$ time to find an independent set of size at least $(1 - O(1/f(n))$. However, Chiba et al. [90] showed that to obtain approximation ratio 2, the input graph must have at least $2^{2^{400}}$ vertices.

6.1.2 Outer-planar decomposition method

In [12] Baker introduces a general method to obtain approximation schemes for various NP-complete problems on planar graphs. This method is based on decomposing a general planar graph into k-outer planar subgraphs.

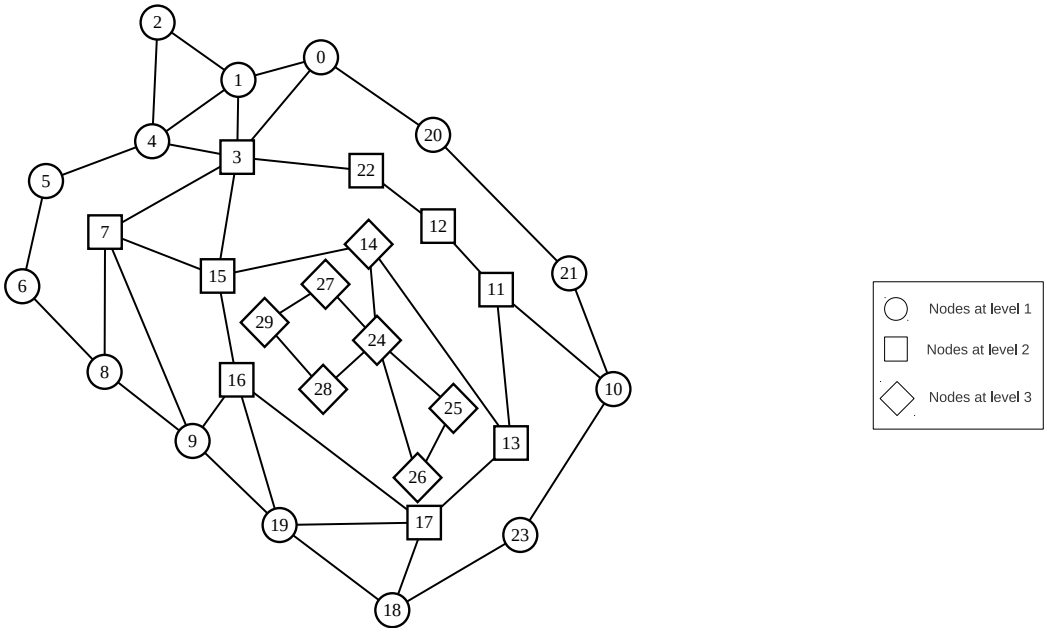


Figure 6.2: A 3-layer outer planar graph

A graph G is called *outer planar* or *1-outer planar* if it has a planar embedding such that all vertices of G are incident to a same face (called outer face). For $k > 1$, G is a *k-outer planar graph*, if it has a planar embedding such that removing the vertices of G on the outer face will result in a $(k-1)$-outer planar graph. Figure 6.2 shows an example of 3-outer planar graph.

Baker shows that the structure of k-outer planar graphs is adaptable with dynamic programming and the optimal solution for the NP-complete problem on k-outer planar graphs can be computed in linear time when k is a constant. For an NP-complete problem P on a general planar graph G, we can generate an approximated solution by decomposing the graph into k-outer or (k+1)-outer planar subgraphs and taking the union of the optimal solutions of each subgraph. The outer-planar decomposition method works as follows:

1. Generate a planar embedding of G using the linear-time algorithm of Hopcroft and Tarjan [71]. Let the embedding has m levels.
2. for $i = 0, 1, ...k - 1$
   (a) Decompose G into subgraphs $G_1, G_2, ...G_r$ as follows:
      - If P is a maximization problem: Remove the vertices of G on levels congruent to $i(\mod k + 1)$. The subgraphs induced by remaining vertices generate disjoint $k-$outer planar subgraphs $G_1, G_2, ...G_r$.
      - If P is a minimization problem: every subgraph G_j, $0 \leq j \leq (m - i - k)/k$, is defined as a subgraph induced by vertices on levels $jk + i$ to $(j + 1)k + i$. Every subgraph G_h is a $k + 1-$outer planar graph that has common vertices with G_{h+1} on level $(h + 1)k + i$.

   (b) Apply a linear-time dynamic programming algorithm to find an optimal solution for P on each subgraph G_h, $1 \leq h \leq r$.

   (c) Generate an approximated solution for problem P on the general graph G by taking the union of the solutions. Let A_i be the approximated solution.
3. If P is a minimization problem the algorithm returns $\min_{i=0,1,...,k-1}\{A_i\}$ and if P is a minimization problem the result will be $\max_{i=0,1,...,k-1}\{A_i\}$.

Baker shows that for general planar graphs if the problem P is a maximization problem, such as maximum independent set, for each fixed k this technique gives a linear-time algorithm that produces a solution whose size is at least $k/(k+1)$ times the optimal solution. If the problem is a minimization problem, such as minimum vertex cover, for each k, it gives a linear-time algorithm that produces a solution whose size is at most $(k + 1)/k$ times the optimal solution. This method resolves the two disadvantages of the separator method. In

[12] some NP-complete problems are listed which are solvable in polynomial time on k-outer planar graphs for constant k.

Planar maximum independent set:

In [12], the outer-planar decomposition method is applied to generate an approximated solution for the planar maximum independent set. It is shown that for every $k-$outer planar subgraph, G_i, an optimal solution, I_i can be computed in linear time by dynamic programming. Taking union of these optimal solution, $I = I_1 \cup I_2 \cup ... \cup I_h$, generate an approximated independent set of size at least $k/(k+1)$ times optimal.

6.2 PTAS for the PLANAR DOMINATING SET problem

Our computational results in Chapter 4 show that branch-decomposition based algorithms may not be practical for solving the DOMINATING SET problem on planar graphs with large branchwidth. Thus, we need to restore the approximation algorithms for solving the DOMINATING SET problem on planar graphs with large branchwidth. In what follows we introduce a PTAS for the PLANAR DOMINATING SET problem based on the outer-planar decomposition method.

6.2.1 Algorithm

It is reported in [12] that a number of NP-hard problems in $k-$outer planar graphs can be solved in polynomial time for constant k and these problems admit a PTAS for general planar graphs. One of such problems is the PLANAR DOMINATING SET problem. It is briefly mentioned in [12] that a PTAS can be obtained by the outer-planar decomposition approach for the PLANAR DOMINATING SET problem. The suggested approximation ratio for minimization problems is $\frac{(k+1)}{k}$. However, as shown below, this approximation ratio is not correct for PLANAR DOMINATING SET and a modification of outer-planar decomposition method is require to get the PTAS for the PLANAR DOMINATING SET problem with approximation ratio $\frac{(k+2)}{k}$.

For minimization problems, outer-planar decomposition method decomposes the planar graph G, into $(k+1)-$outer planar subgraphs. For every i, $0 \leq i < k$, the optimal solution is computed for $(k+1)-$outer planar subgraph induced by levels $jk+i$ to $(j+1)k+i$, $j \geq 0$ and the union of these solutions generates an approximated solution. The outer-planar decomposition method returns the smallest approximated solution. For example, it

is proved that the approximated solution is at most $\frac{(k+1)}{k}$ times the optimal for the minimum vertex cover problem. The proof is as follows. For a given graph G and an integer k, let C be an optimal solution and C_a be the approximated solution computed by the outer-planar decomposition method. Clearly, for some $0 \leq t < k$, there are at most $|C|/k$ vertices on levels congruent to $t(\mod k)$. For every $j \geq 0$, $G_{(t,j)}$ is defined as a subgraph induced by levels $jk+t$ to $(j+1)k+t$. Let $C_{(t,j)}$ be the set of vertices of C in $G_{(t,j)}$, and $C'_{(t,j)}$ be a minimum vertex cover of $G_{(t,j)}$ computed by the algorithm. Clearly, for every $j \geq 0$ we have $|C'_{(t,j)}| \leq |C_{(t,j)}|$. Therefore,

$$|C_a| \leq \sum_{j \geq 0} |C'_{(t,j)}| \leq \sum_{j \geq 0} |C_{(t,j)}|$$

Each subgraph $G_{(t,j)}$ overlaps with $G_{(t,(j+1))}$ on level $(j+1)k+t$. Thus, the vertices of C on levels congruent to $t(\mod k)$ are counted twice in $\sum_{j \geq 0} |C_{(t,j)}|$, therefore $\sum_{j \geq 0} |C_{(t,j)}| \leq \frac{(k+1)}{k}|C|$.

In [12], using the same proof, it is suggested that for the PLANAR DOMINATING SET problem the approximation ratio is $\frac{(k+1)}{k}$. However, in what follows we give a counter example that this proof is not correct for the PLANAR DOMINATING SET problem.

Let G be a plane graph shown in Figure 6.3a with 46 vertices and $k = 2$. A minimum dominating set D of G, containing 9 vertices is shown by black vertices. G has 6 levels and the outer level is numbered one. For $t = 0$ there are only three vertices of D on levels congruent to 0, levels 2,4,6. For every subgraph $G_{(0,j)}$, let $D_{(0,j)}$ be a subset of D in $G_{(0,j)}$, such that the vertices on levels $2j$ and $2j+1$ are dominated by $D_{(0,j)}$. Let $D'_{(0,j)}$, $j \geq 0$, be a minimum general dominating set of $G_{(0,j)}$, that dominates the vertices on levels $2j$ and $2j+1$. For $j = 2$, $G_{(0,2)}$ and $D_{(0,2)}$ are shown in Figure 6.3b and the vertices of $D'_{(0,2)}$ are shown with bold lines in Figure 6.3c. For this subgraph, $D'_{(0,2)} \leq D_{(0,2)}$ does not hold, and thus the above proof is not valid for the PLANAR DOMINATING SET problem.

Decomposing the input graph G, into $(k+1)-$outer planar subgraphs $G_{(t,j)}$, is due the fact that the vertices of level $(j+1)k+t-1$ can be dominated by the vertices on level $(j+1)k+t$. However, this assumption is not enough since the vertices on level $jk+t$, can also be dominated by the vertices on level $jk+t-1$. Therefore, in the outer-planar decomposition method we include the vertices on level $jk+t-1$ in the subgraph $G_{(t,j)}$.

Therefore, we modify the outer-planar decomposition method to decompose the planar graph into $k+2-$outer planar subgraphs. For every t, $0 \leq t < k$, the minimum general

dominating set is computed for $k+2-$outer planar subgraph induced by levels $jk+t-1$ to $(j+1)k+t$, $j \geq 0$.

For a $(k+2)-$ outer planar subgraph, induced by levels $m-1$ to $m+k$, let U be the set of vertices on levels $m-1$ and $m+k$, and V be the set of vertices of the remaining levels. A minimum general dominating set D_g is a minimum set of vertices in level $m-1$ to $m+k$, such that every vertex in levels m to $m+k-1$ is dominated by D_g. Figure 6.4 shows a general dominating set computed for a $k+2-$outer planar subgraph $G_{(0,2)}$.

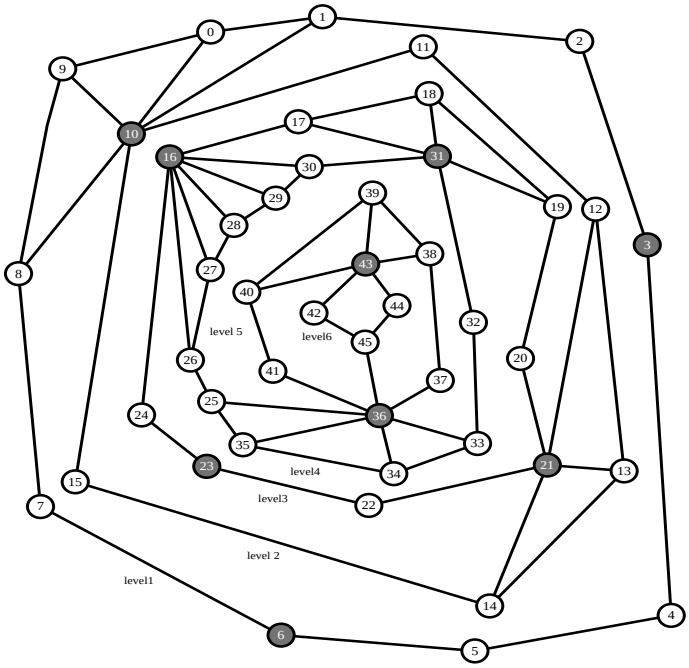
Let G be a planar graph with m levels, $1, ..., m$, and G be decomposed into $(k+2)-$outer planar subgraphs, $G_1, G_2, ..., G_r$. For every subgraph G_i, $2 \leq i \leq r-1$, we calculate a minimum $D_g(G_i)$. For G_1 we need to calculate a minimum $D_g(G_1)$ such that the vertices in levels 1 to k are dominated. For G_r we calculate a minimum $D_g(G_r)$ such that the vertices in levels $m-k$ to m are dominated. Therefore, the dominating sets for G_1 and G_r includes the vertices that ensures the vertices in levels 1 and m are also dominated.

In order to calculate the optimal dominating set of every subgraph we use the FT algorithm introduced in Chapter 4. Clearly, the branchwidth of every k-outer planar graph is at most $2k$, therefore by choosing a small k we can generate the subgraphs with small branchwidth and find their optimal solutions in practical time and memory space. Through applying the FT algorithm we calculate an optimal general dominating set for every subgraph. The union of these optimal dominating sets generates an approximated solution for G.

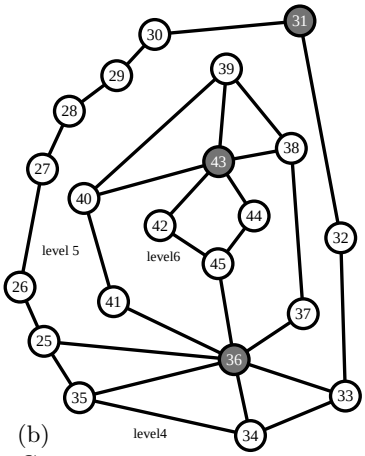
6.2.2 Approximation ratio

For a given planar graph $G(V, E)$ and a positive integer k, the algorithm described in the previous section works as follows: For each $i, 0 \leq i < k$, it uses the FT algorithm to find the optimal solutions for the $(k+2)-$outer planar subgraphs induced by levels $jk+i-1$ to $(j+1)k+i$, $j \geq 0$. For each i, the union of the solutions gives a dominating set of G. The algorithm picks the minimum one from these solutions, called D'.

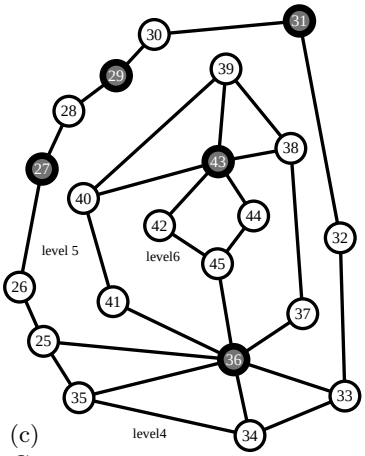
In what follows, we show that D' is a dominating set of G and $|D'| \leq \frac{(k+2)}{k}.opt$. Let D be the minimum dominating set of G. Let D_{ij} be the set of vertices of D in levels $jk+i-1$ through $(j+1)k+i$, and D'_{ij} the optimal dominating set computed by PTAS for the subgraph induced by these levels. Since PTAS computes an optimal dominating set for every subgraph $|D'_{ij}| \leq |D_{ij}|$. Clearly, for some $t, 0 \leq t < k$, at most $2|D|/k$ vertices in D are in levels congruent to $t \mod k$ or $(t+1) \mod k$. Thus, we have



(a) A plane graph G and its minimum dominating set



(b) $G_{(0,2)}$ and $D_{(0,2)}$



(c) $G_{(0,2)}$ and $D'_{(0,2)}$

Figure 6.3: A counter example such that $|D_{(0,2)}| \leq |D'_{(0,2)}|$.

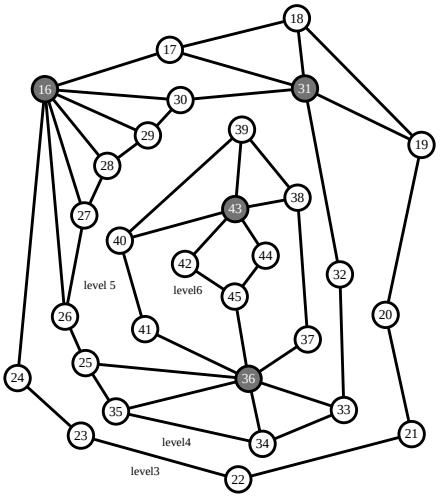


Figure 6.4: a minimum general dominating set for (k+2)-outer planar subgraph $G_{(0,2)}$

$|D'| = |D'_{t0}| + |D'_{t1}| + |D'_{t2}| + ... + |D'_{tr}| \leq |D_{t0}| + |D_{t1}| + ... + |D_{tr}|$

Since the subgraphs have overlaps, the vertices of D in the levels congruent to $t \mod k$ and the levels congruent to $(t+1) \mod k$ are counted twice. Therefore, $|D'| \leq ((k+2)/k)|D|$.

6.3 Computational study of PTAS for the PLANAR DOMINATING SET problem

Our computational results in Chapter 4 show that the FT algorithm is not practical on our test platform for graphs with branchwidth greater than 12. For these graphs, we study the performance of the PTAS introduced in the previous section. The PTAS solution is implemented in C++ and its performance is tested on 4 different classes of graphs including the Delaunay triangulations of point sets taken from TSPLIB [95], triangulations and intersection graphs generated by LEDA and Gabriel graphs generated using the points uniformly distributed in a two-dimensional plane. These classes are the classes (2),(3),(4) and (5), respectively, in the computational study of the FT algorithm in Chapter 4. The same computing platform is used for the study of these classes. Table 6.1 shows the computational results of the PTAS for the PLANAR DOMINATING SET problem. For every instance we calculate the approximated solutions for three different values of k, 2,3 and 4, and for every value of k we calculate all possible decomposing the graph into $(k+2)$-outer planar components. We choose the best value for an approximated solution. For some instances

with small branchwidth, we also include the results of the FT algorithm. The minimum dominating set of graph G, computed by the exact algorithm is indicated by $\gamma(G)$ in Table 6.1, and for every value of k, $D_{PTAS}(G)$ is the size of dominating set computed by the approximation algorithm. The running time is in seconds.

In order to compare the approximated dominating sets obtained from the PTAS with the optimal solutions, we include some instances with small branchwidth for every class of graphs, such that a minimum dominating set can be computed using the FT algorithm. The Exact Alg. column shows the results of the FT algorithm. We use three values for k to decompose the instances into $k+2-$outer planar component. Clearly, the branchwidth of every $(k+2)-$outer planar graph is at most $2k+4$. Hence, by increasing k the size of subgraphs and their branchwidth will increase. Theoretical results suggest that increasing k gives smaller approximated solutions for the maximization problems. Our computing results confirms the theoretical analysis of the outer-planar decomposition method. For example for $k=4$ every instance can be decomposed into subgraphs with branchwidth at most 12. This is the largest value of branchwidth that can be processed on our computational platform.

	Graph	$\|E(G)\|$	bw	Exact Alg.			k=2			k=3			k=4		
	G			$\gamma(G)$	l	time	D_{PTAS}	l	time	D_{PTAS}	l	time	D_{PTAS}	l	time
(2)	kroB150	436	10	23	10	10	33	8	1.94	28	8	2.07	-	-	-
	pr299	864	11	47	11	37	66	8	3.15	56	10	11.42	-	-	-
	tsp225	622	12	37	12	110	50	8	3.55	46	9	5.21	-	-	-
	a280	788	13	43	13	337	60	8	2.85	53	10	8.40	51	12	12.09
	rd400	1183	17	-	-	-	83	8	8.25	75	10	35.30	74	12	351.93
	pcb442	1286	17	-	-	-	82	8	5.62	79	10	10.46	78	10	10.86
	d657	1958	22	-	-	-	132	8	12.83	123	10	64.89	120	12	604.10
	pr1002	2972	21	-	-	-	210	8	45.31	190	10	115.65	182	12	1253.9
(3)	tri2000	5977	8	321	7	198	379	7	109.87	361	7	175.59	-	-	-
	tri4000	11969	9	653	7	1903	763	7	505.34	724	7	733.06	-	-	-
	tri6000	17979	9	975	8	3576	1187	8	826.56	1136	8	1994.53	-	-	-
	tri8000	23975	9	1283	7	7750	1526	7	1465.7	1430	7	2858.63	-	-	-
	tri10000	29976	9	1606	7	16495	1901	7	4063.06	1804	7	4977.06	-	-	-
	tri11000	32972	14	-	-	-	2116	7	4251.29	1987	8	5910.8	1958	8	12341.1
	tri12000	35974	14	-	-	-	2279	7	3950.5	2164	7	5370.18	2132	7	6865.08
	tri14000	41974	15	-	-	-	2678	7	8405.4	2514	7	8220.49	2434	7	9208.72
	tri16000	47969	16	-	-	-	3064	7	9413.42	2920	7	10060.1	2885	7	12794.4
(4)	rand6000	10293	11	1563	9	150	1715	8	113.84	1658	8	104.85	-	-	-
	rand10000	17578	13	2535	10	869	2850	8	515.75	2721	8	535.87	2692	9	432.23
	rand15000	26717	14	3758	12	2769	4288	8	2002.03	4144	10	1313.14	-	-	-
	rand16000	28624	13	4002	13	5917	4584	8	2247.65	4379	10	2443.27	4295	11	2027.7
	rand20000	35975	14	4963	14	13993	5688	8	4483.16	5465	10	4241.65	5368	12	5017.02
	rand25000	40378	16	-	-	-	7101	8	11693.4	6731	10	6407.91	6632	12	9470
(5)	Gab500	949	13	115	12	238	149	8	7.45	136	10	18.02	129	10	18.95
	Gab600	1174	14	135	14	3074	172	8	11.57	164	10	26.05	156	10	22.10
	Gab700	1302	14	162	14	5710	199	8	11.72	187	10	22.81	183	10	24.30
	Gab800	1533	17	-	-	-	235	8	16.90	225	10	51.82	205	12	24.30
	Gab900	1758	17	-	-	-	256	8	22.52	243	10	48.39	231	12	344.50
	Gab1000	1901	18	-	-	-	292	8	25.88	260	10	49.69	259	12	781.89
	Gab1500	2870	21	-	-	-	436	8	48.98	402	10	116.37	385	12	960.71

Table 6.1: Computational results (time in seconds) of PTAS for the PLANAR DOMINATING SET problem.

Chapter 7

Practical performances of the PLANAR DOMINATING SET algorithms

Since the theory of NP-completeness has reduced hopes that NP-hard problems can be solved in polynomial time, heuristic and approximation algorithms have attracted more attentions. These algorithms compute near optimal solutions within a reasonable time for problems of realistic size and complexity. In this chapter we compare the performance of PTAS with the performance of four different heuristic algorithms introduced in [99] for PLANAR DOMINATING SET problem. In what follows we briefly explain these heuristic algorithms (for more details refer to [99]).

7.1 Heuristic methods

In [99], six heuristic algorithms for the DOMINATING SET problem are introduced. We studied the performance of these six methods, but because two of them were very poor in performance we report only four algorithms in [99].

Let D be the dominating set computed using these algorithms, initially D is empty unless it is assigned.

Greedy: In each iteration of this algorithm the vertex with maximum number of uncovered neighbours is added to D. If there are more than one vertex with the same uncovered

neighbours, one of vertices is chosen at random.

Greedy-Rev: Unlike the Greedy algorithm, set D initially contains all the vertices of the input graph. In each iteration, a vertex is removed from D, such that the resulting set remains a dominating set of G. A vertex is chosen to be removed, by ordering the vertices of D in increasing degree, and removing the first vertex that does not dominate any vertex uniquely.

Greedy-Ran: This algorithm differs with the Greedy algorithm only in the method for choosing a vertex to include in D. It randomly includes a vertex at each step based on an assigned probability value. The probability value of a vertex depends on the number of its uncovered neighbours.

Greedy-Vote: This algorithm does not include a vertex u in D only based on the number of vertices which are covered by u. It rather pays attention to the vertices which are covered by u and check whether they can be covered in some alternate way.

7.2 Computational study of PTAS and heuristic methods

We study the performances of the above algorithms for the four classes of planar graphs that are used in the study of the PTAS in the previous chapter. These algorithms are implemented in C++. Table 7.1 shows the computational results of these heuristic algorithms and PTAS.

In Table 7.1, D_{Gr}, D_{Rev}, D_{Ran} and D_{Vot} are the sizes of dominating sets computed by the heuristic method Greedy, Greedy-Rev, Greedy-Ran and Greedy-Vote respectively. For every graph instance . If the size of the instance allows the application of FT algorithm, we include the size of the minimum dominating set of the instance as well. For the PTAS we include the best result, D_{PTAS} for every instance from Table 6.1. As the results in Table 7.1 show the heuristic methods are always faster than PTAS, however the size of dominating sets computed by these methods are significantly larger than the size of the dominating sets computed by PTAS for most of instances.

Based on our computational results, Greedy algorithm resulted the smallest dominating sets compare to other heuristic algorithms. Table 7.2 shows the results of our computational study for the exact algorithm, Greedy (the best Heuristic method) and PTAS for graph instances whose branchwidths are small enough to run the FT algorithm. Since the branchwidth of graphs in Class(2) grows fast in size of the graph, we have only included

	Graph G	$\|E(G)\|$	$\gamma(G)$	Greedy Alg.		Greedy-Rev Alg		Greedy-Ran Alg		Greedy-Vote Alg		PTAS	
				D_{Gr}	time	D_{Rev}	time	D_{Ran}	time	D_{Vot}	time	D_{PTAS}	time
(2)	kroB150	436	23	27	0.002	31	0.01	40	0.006	31	0.002	28	2.08
	pr299	864	47	54	0.003	63	0.032	76	0.019	62	0.005	56	11.42
	tsp225	622	37	49	0.153	54	0.02	58	0.011	50	0.003	46	5.21
	a280	788	43	51	0.004	62	0.025	71	0.019	62	0.006	51	12.09
	rd400	1183	-	78	0.007	92	0.032	102	0.032	90	0.009	74	351.93
	pcb442	1286	-	76	0.908	90	0.063	122	0.038	87	0.01	78	10.86
	d657	1958	-	126	0.016	146	0.128	175	0.084	143	0.021	120	604.10
	pr1002	2972	-	190	0.032	236	0.328	263	0.214	194	0.04	182	1253.9
(3)	tri2000	5977	321	365	0.116	379	1.119	519	0.877	464	0.168	361	175.59
	tri4000	11969	653	729	0.183	765	1.792	1031	1.462	787	0.544	724	733.06
	tri6000	17979	975	1118	0.418	1166	4.14	1567	2.721	1306	0.541	1136	1994.53
	tri8000	23975	1283	1449	0.715	1522	7.003	2102	5.815	1653	0.918	1430	2858.63
	tri10000	29976	1606	1819	1.117	1906	11.524	2597	7.263	2302	1.572	1804	4977.06
	tri11000	32972	-	2040	1.375	2116	14.092	2874	10.669	3431	2.561	1958	12341.1
	tri12000	35974	-	2186	1.607	2278	16.538	3119	10.836	2741	2.243	2132	6865.08
	tri14000	41974		2576	2.462	2664	22.976	3654	17.757	3317	3.163	2434	9208.72
	tri16000	47969	-	2917	2.839	3033	30.694	4161	21.197	3684	4.005	2885	12794.4
(4)	rand6000	10293	1563	1932	0.748	2166	4.517	2332	3.474	2908	1.206	1658	104.85
	rand10000	17578	2535	3197	2.06	3618	13.33	3834	11.168	4164	2.878	2692	432.23
	rand15000	26717	3758	4698	4.861	5402	29.487	5687	23.409	7277	7.641	4144	1313.14
	rand16000	28624	4002*	5039	5.176	5744	35.589	5985	28.416	7552	10.327	4295	2027.7
	rand20000	35975	4963*	6273	8.053	7168	55.948	7513	46.834	8571	11.903	5398	5017.02
	rand25000	45327	-	7772	12.467	8942	91.039	9358	72.038	11865	20.615	6632	9470
(5)	Gab500	949	115	146	0.006	173	0.039	176	0.022	160	0.007	129	18.95
	Gab600	1174	135	168	0.007	199	0.051	210	0.044	171	0.009	156	22.10
	Gab700	1302	162	200	0.01	242	0.072	258	0.046	238	0.012	183	24.30
	Gab800	1533	-	227	0.012	270	0.097	281	0.062	307	0.019	205	24.30
	Gab900	1758	-	254	0.016	303	0.103	327	0.091	323	0.022	231	344.50
	Gab1000	1901	-	280	0.019	344	0.146	370	0.106	423	0.03	259	781.89
	Gab1500	2870	-	426	0.042	507	0.335	537	0.268	496	0.051	385	960.71

Table 7.1: Computational results for heuristic methods and PTAS for the PDS problem (time in second).

	Graph	$\|E(G)\|$	Exact Alg.		Greedy Alg		PTAS	
	G		$\gamma(G)$	time	D_G	time	D_{PTAS}	time
(2)	kroB150	436	23	10	27	0.002	28	2.08
	pr299	864	47	37	54	0.032	56	11.42
	tsp225	622	37	110	49	0.153	46	5.21
	a280	788	43	337	51	0.004	51	12.09
(3)	tri2000	5977	321	198	365	0.116	361	175.59
	tri4000	11969	653	1903	729	0.183	724	733.06
	tri6000	17979	975	3576	1118	0.418	1136	1994.53
	tri8000	23975	1283	7750	1449	0.715	1430	2858.63
	tri10000	29976	1606	16495	1819	1.117	1804	4977.06
(4)	rand6000	10293	1563	150	1932	0.748	1658	104.85
	rand10000	17578	2535	869	3197	2.06	2692	432.23
	rand15000	26727	3758	2769	4698	4.861	4144	1313.14
	rand16000	28624	4002*	5917	5039	5.176	4295	2027.7
	rand20000	35975	4963*	13993	6273	8.053	5398	5017.02
(5)	Gab500	949	115	238	146	0.006	129	18.95
	Gab600	1174	135*	3074	168	0.007	156	22.10
	Gab700	1302	162*	5710	200	0.01	183	24.30

Table 7.2: Computational results for Exact, Greedy and PTAS algorithms for small instances (time in second).

small instance of this class in Table 7.1. The running time of exact algorithm is not very large, for this class of graphs, if the graph branchwidth is smaller than 14, we suggest the exact algorithm. For the instances of Class (3), the FT algorithm is extremely slow. If the running time is the driving factor, we suggest the Greedy algorithm for this class of graphs. As the results in Table 7.2 suggest the size of dominating sets computed by Greedy is considerably bigger than PTAS results. For instances of Classes (4) and (5) the exact algorithms is very time consuming. However, the differences between the results of PTAS and Greedy is significant enough to tolerate the longer running time of the PTAS. For instance, for the rand20000 with 35975 edges, the FT algorithm take almost four hours to compute the size of an optimal dominating set (not the vertices of dominating set), however PTAS computes a dominating set including the vertices in dominating set in less than two hours.

Table7.3 shows the computational results for the instances that FT is not able to find an optimal solution in practical time and memory. The computational results show that for all of these instances except one instance, the D_{PTAS} is smaller than D_{Gr}. In summary for applications with running time priority, Greedy is a better choice to compute an approximated dominating set, and if the running time is not a big concern, PTAS is a better option for instances whose optimal dominating set cannot be computed using the FT algorithm.

	Graph G	$\|E(G)\|$	Greedy Alg.		PTAS	
			D_{Gr}	time	D_{PTAS}	time
(2)	rd400	1183	78	0.007	74	351.93
	pcb442	1286	76	0.908	78	10.86
	d657	1958	126	0.016	120	604.10
	pr1002	2972	190	0.032	182	1253.9
(3)	tri11000	32972	2040	1.375	1958	12341.1
	tri12000	35974	2186	1.607	2132	6865.08
	tri14000	41974	2576	2.462	2434	9208.72
	tri16000	47969	2917	2.839	2885	12794.4
(4)	rand25000	45327	7772	12.467	6632	9470
(5)	Gab800	1533	227	0.012	205	24.30
	Gab900	1758	254	0.016	231	344.50
	Gab1000	1901	280	0.019	259	781.89
	Gab1500	2870	426	0.042	385	960.71

Table 7.3: Computational results for Greedy and PTAS for large instances (time in second).

Chapter 8

Conclusion and future works

8.1 Conclusion

In this dissertation we study various exact and approximate algorithms for solving PLANAR DOMINATING SET problems. The class of exact and approximation algorithms we undertake in our study are branch-decomposition based (i.e., they use the branch-decomposition as a method to decompose the graph into smaller subgraphs). While finding the branchwidth and an optimal branch-decomposition of planar graphs is shown to be solvable in polynomial time, our studies show that previous implementations are impractical when processing graphs over a few thousand edges on our test platform (a Linux box with 2 giga bytes of internal memory). Thus, as a first step we develop an improved implementation of Seymour and Thomas procedure that, given an integer β, decides whether a planar graph G has branchwidth at least β or not. Our implementation is able to compute the branchwidth for graphs of up to one hundred thousand edges in reasonable time using the same test platform. This suggests that the required memory may not be a bottleneck for computing branchwidth and optimal branch-decompositions of planar graphs in practice.

We incorporated this efficient implementation of ST Procedure into the edge contraction method of Seymour and Thomas [102], and Gu and Tamaki's method [61] to develop a memory efficient optimal branch-decomposition tool. Additional path simplification procedure is utilized for a class of graphs called PIGALE graphs to achieve better efficiency. These improvements lead to a practical tool for evaluating the performance of FPT algorithms on planar graphs. We use this tool to evaluate Fomin and Thilikos's PLANAR DOMINATING SET algorithm [56] on a wide range of planar graphs. The computational results coincide

with the theoretical analysis of the algorithm, that is, it is efficient for graphs with small branchwidth but may not be practical for graphs with large branchwidth. More specifically, using a computational platform with 2.4 GHz CPUs and 3 giga bytes of internal memory, we are able to find a minimum dominating set (resp. the dominating number) for graphs with kernel branchwidth at most 13 (resp. 14) in a few hours. Our results confirm that this method however is indeed not practical for graphs with larger branchwidth. Thus, we resort to approximation methods for processing graphs with larger branchwidths. We incorporated our branch-decomposition procedure into Baker's outer planar decomposition method and developed a practical PTAS for processing graphs with large branchwidths. We completed our computational study by comparing the performance of FT algorithms, PTAS and well-known existing heuristics for PLANAR DOMINATING SET problems. Our results show that, expectedly, heuristic methods are faster than exact branch-decomposition based and PTAS methods. However, the heuristic methods generate dominating sets that are on average 20% and 5% larger than the exact branch-decomposition based and PTAS methods, respectively. Therefore, if time is not a big concern the methods of choice are exact branch-decomposition based and PTAS. For instance, while an optimal solution for graphs with branchwidth less than 14 can be computed in a few hours, it can be up to 32% smaller than the solution computed by a heuristic. For the graphs that the exact method is impractical, the choice of PTAS versus heuristic is instance dependant. For example for our tested Gabriel graphs PTAS generates solutions up to 40% smaller than those generated by the heuristic.

We also contributed in the analysis of PLANAR CDS problem that is a variation of the DOMINATING SET problem in which the dominating set has to induce a connected graph. Although the CDS problem seems very close to the DOMINATING SET problem, the solutions to the later cannot be applied to the CDS problem in a straightforward manner. A new branch-decomposition based algorithm for the CDS problem is introduced by Dorn and et. al. [43] based on a novel frame work on the geometric properties of branch-decomposition of planar graphs and non-crossing partitions. They suggest that using this framework the PLANAR CDS problem can be solved in $O(2^{O(bw(G))}n+n^3)$. In this dissertation we provide the the details of the DPBF algorithm based on the framework introduced in [43]. We also analyze the running time of the DPBF algorithm. With a more careful analysis, we improve the upper bound of the algorithm. We also prove an improved upper bound for the branchwidth of the graph in term of the connected dominating number of the graph.

Using this upper bound we improve the running time of FPT algorithm for PLANAR CDS problem as well. We evaluate the performance of DPBF Algorithm for the CDS problem on a wide range of planar graphs. The computational results in this case coincide with the theoretical analysis of the algorithm as well. Using a computer with a CPU of 2.4GHz and 3G Bytes memory space, it is possible to find a minimum CDS for graphs with the kernel branchwidths of at most 10 in a few hours. Since the branchwidth of a planar graph can be computed in $O(n^2 \log n)$ time by the $O(n^2)$ time rat-catching algorithm and a binary search, one may first find the branchwidth of the input graph and then decide if DPBF algorithm is applicable based on the results presented in this part of thesis as a guideline.

Our computational study provides a guideline for using the branch-decomposition based algorithms to solve important local and non-local problems in planar graphs. We show that the PLANAR DOMINATING SET and CDS problems can be solved in practice for a wide range of graphs. Our results show that PLANAR DOMINATING SET problem can be solved using branch-decomposition based algorithms for graphs with branchwidth at most 14 on our testing platform. Branch-decomposition based algorithms are not practical for planar graphs with large branchwidth. For instance, we were not able to solve graphs with bandwidth larger than 14 using our testing platform due to memory constraint. For these graphs, when the longer running times can be tolerated, PTAS is the method of choice, otherwise heuristics are the best options. This work, in addition, provides a tool for computing the optimal connected dominating set of planar graphs and may bring the sphere-cut decomposition and non-crossing partitions based approach closer to practice. Our computational results show that the sphere-cut decomposition based algorithm is practical for solving PLANAR CDS problem for graphs with branchwidth up to 10 using a common personal computer.

8.2 Future works

Many interesting open questions arise in the realm of branch-decomposition based algorithms on planar graph problems. Here we discuss some of these problems related to our work. The improvements we introduced for the implementations of ST Procedure in chapter 3, are concentrated on reducing memory usage. An interesting followup to this aspect is to work on running time improvements for this procedure. One possible way is to find an efficient starting phase that leads to a conclusion in a lesser time. Note that this improvement

is only effective when the ST Procedure returns false (i.e. rat-catcher wins).

The time and memory requirement for the FT and DPBF algorithms exponentially depends on the branchwidth $bw(H)$ of a kernel H of the input graph. Thus, it is worth to develop more powerful data reduction rules to reduce $bw(H)$. It is also worth to develop heuristics to reduce $l(H)$, i.e the maximum number of grey vertices in $\partial(A_e)$ for every link in the used branch-decomposition. Those heuristics should provide solutions close to the optima in a fraction of cost of FT Algorithm for graphs with large branchwidth.

Another interesting and challenging problem is to develop a practical fast matrix multiplication method. Fast matrix multiplications and distance products have a huge impact on the efficiency of the branch-decomposition based algorithms. However, as discussed in Chapters 4 and 5, the existing fast matrix multiplication methods are impractical. Introducing a practical fast matrix multiplication can improve the running time of FT and DPBF algorithms significantly.

It is known that the PLANAR CDS problem admits PTAS. The approach for the PTAS is to partition an input graph into subgraphs of fixed branchwidth, find a minimum CDS for each subgraph and combining the solutions of subgraphs into a solution of the input graph. An interesting approach is to utilize the DPBF Algorithm for finding subgraph solutions that will result in a more efficient PTAS for the graphs with large branchwidth.

Bibliography

[1] Library of efficient data types and algorithms, version 5.2. *http://www.algorithm-solutions.com/enleda.htm*, 2008.

[2] Public implementation of a graph algorithm library and editor. *http://pigale.sourceforge.net/*, 2008.

[3] J. Alber, N. Betzler, and R. Niedermeier. Experiments on data reduction for optimal domination in networks. *Journal of Annals of operations research*, 146(1):105–117, 2006.

[4] J. Alber, F. Dorn, and R. Niedermeier. Experiments on optimally solving np-complete problems on planar graphs, manuscript. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.4973*, 2001.

[5] J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics*, 145(2):219–231, 2005.

[6] J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics*, 145(2):219–231, 2005.

[7] M.R. Alber, J. abd Fellows and R. Niedermeier. Polynomial time data reduction for dominating set. *Journal of ACM*, 51(3):363–384, 2004.

[8] E. Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[9] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embedding in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[10] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[11] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[12] B.S. Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of ACM*, 41(1):153–180, 1994.

[13] C. Berge. *Graphs and Hypergraphs*. Amerocan Elsevier, 1973.

[14] M. Bhardwaj, S. Misra, and G. Xue. Distributed topology control in wireless ad hoc networks using β-skeletons. *Workshop on High Performance Switching and Routing*, pages 371–375, 2005.

[15] R. Bhatia, S. Khuller, R. Pless, and Y.J. Sussamnn. The full-degree spanning tree problem. *Networks*, 36(4):203–209, 2000.

[16] Z. Bian. *Algorithms for wavelenght assignment and call control in optical networks*. PhD thesis, Simon fraser university, 2008.

[17] Z. Bian and Q. Gu. Computing branch decomposition of large planar graphs. In *the 7th International Workshop on Experimental Algorithms (WEA'08)*, pages 87–100, 2008.

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

[19] J. Blum, M. Ding, M. Thaeler, and X. Cheng. Connected dominating set in sensor networks and manets. *Handbooks of Combinatorial Optimization*, pages 329–369, 2004.

[20] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[21] H.L. Bodlaender. A linear time algorithm for finding tree-decomposition of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[22] H.L. Bodlaender. Treewidth: characterizations, applications, and computations. In *32nd International Workshop on Graph Theoretical Concepts in Computer Science (WG2006)*, pages 1–14, 2006.

[23] H.L. Bodlaender, A. Brandstadt, D. Kratsch, M. Rao, and J. Spinrad. On algorithms for (p5,gem)-free graphs. *Theoretical computer science*, 349(1):2–21, 2005.

[24] H.L. Bodlaender, A. Grigoriev, and Koster A.M.C.A. Treewidth lower bounds with brambles. *Algorithmica*, 51(1):81–98, 2008.

[25] H.L. Bodlaender and K. Jansen. On the complexity of the maximum cut problem. *Nordic Journal of Computing*, 7:14–31, 2000.

[26] H.L. Bodlaender and U. Rotics. Computing the treewidth and the minimum fill-in with the modular decomposition. *Algorithmica*, 36:375–408, 2003.

[27] H.L. Bodlaender and D. Thilikos. Constructive linear time algorithm for branchwidth. *In Proc. of 24th International Colloquium on Automata, Languages, and Programming, ICALP97*, pages 627–637, 1997.

[28] I. Cahit. Face labeling of maximal planar graphs. *Applied mathematics E-Notes*, 11:1–11, 2011.

[29] X. Cheng, M. Ding, H. Du, and X. Jia. Virtual backbone construction in multihop ad hoc wireless networks. *Wireless communications and mobile computing*, 6(2):183–190, 2006.

[30] M. Chlebik and J. Chlebikova. Approximation hardness of edge dominating set problems. *Journal of Combinatorial Optimization*, 11(3):279–290, 2006.

[31] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[32] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[33] E.D. Demaine, F. V. Fomin, M.T. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM Journal on Discrete Mathematics*, 18:501–511, 2004.

[34] E.D. Demaine, F. V. Fomin, M.T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k,r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1:33–47, 2005.

[35] E.D. Demaine, F. V. Fomin, M.T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and h-minor free graphs. *Journal of the ACM*, 52:866–893, 2005.

[36] E.D. Demaine and M.T. Hajiaghayi. Bidimensionality: new connections between fpt algorithms and ptas. pages 590–601, 2005.

[37] E.D. Demaine and M.T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Computer Journal*, 51(3):292–302, 2008.

[38] E.D. Demaine, M.T. Hajiaghayi, and D. M. Thilikos. Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single crossing graphs as minors. *Algorithmica*, 41:245–267, 2005.

[39] F. Dorn. Tuning algorithms for hard planar graph problems. Master's thesis, University at Tubingen.

[40] F. Dorn. Dynamic programming and fast matrix multiplication. In *the 14th Annual European Symposium on Algorithms, ESA2006*, volume 14, pages 280–291. Springer-Verlag, 2006.

[41] F. Dorn. How to use planarity efficiently: new tree-decomposition based algorithms. In *the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG2007)*, volume 4769, pages 280–291, 2006.

[42] F. Dorn, E. Penninkx, H.L. Bodlaender, and F.V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut branch decompositions. In *the 13th Annual European Symposium on Algorithms (ESA2005)*, volume 3669, pages 95–106, 2005.

[43] F. Dorn, E. Penninkx, H.L. Bodlaender, and F.V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut branch decompositions. *Technical report, UU-CS-2006-006, Department of Information and Computing Science*, 2006.

[44] M.R. Downey, R.G.and Fellow. Fixed parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.

[45] R.G. Downey and M.R. Fellow. *Parameterized complexity(Monographs in Computer Science)*. Springer, 1999.

[46] W. Espelage, F. Gurski, and E. Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In *the 27th Workshop on Graph Theoretical Concepts in computer science (WG 2001)*, pages 117–128, 2001.

[47] U. Feige, M.T. Hajiaghayi, and J.R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM journal of computing*, 38(2):629–657, 2008.

[48] M. Fellows, H. Fernau, J. Alber, H. Fan, and R. Niedermeier. Refined search tree technique for dominating set on planar graphs. In *the 26th Mathematical Foundations of Computer Science (MFCS2001)*, volume 2136, pages 111–122, 2001.

[49] M.R. Fellows, J. Alber, and R. Niedermeier. Polynomial time data reduction for dominating set. *Journal of the ACM*, 51(3):363–384, 2004.

[50] H. Fernau, T. Kloks, J. Alber, H.L. Bodlaender, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Journal of Graph Theory*, 63:461–493, 2002.

[51] U. Fiege. A threshold of $\ln n$ for approximating set cover. *Journal of ACM*, 45(4):634–652, 1998.

[52] F.V. Fomin, F. Dorn, and D.M. Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. In *the nineteenth annual ACM/SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 631–640, 2008.

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

[54] F.V. Fomin, F. Gandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008.

[55] F.V. Fomin, F. Grandoni, and D. Kratch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bull. EATCS*, 87:47–77, 2005.

[56] F.V. Fomin and D.M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM Journal on computing*, 36(2):281–309, 2006.

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

[58] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica*, 18:25–66, 1967.

[59] M.R. Garey and D.S. Johnson. *Computer and intractability:A Guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[60] M. Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

[61] Q. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $o(n^3)$ time. *ACM Transactions on Algorithms*, 4(3):30:1–30:13, 2008.

[62] Q. Gu and H. Tamaki. Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in $o(n^{1+\epsilon})$ time. *Theoretical Computer Science*, 412:4100–4109, 2011.

[63] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[64] J. K. Gundel. *The Role of Topic and Comment in Linguistic Theory*. PhD thesis, University of Texas, 1974. Published by the Indiana University Linguistics Club, Bloomington, 1977.

[65] M.M. Halldorson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993.

[66] T.W. Haynes, S.T. Hedetniemi, and M.A. Henning. Domination in graphs applied to electronic power networks. *SIAM Journal on discrete mathematics*, 15(4):519–529, 2002.

[67] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of domination in graphs*. Marcel Dekker, 1998.

[68] I.V. Hicks. Planar branch decompositions i: The ratcatcher. *INFORMS Journal on Computing*, 17(4):402–412, 2005.

[69] I.V. Hicks. Planar branch decompositions ii: The cycle method. *INFORMS Journal on Computing*, 17:413–421, 2005.

[70] I.V. Hicks, A.M.C.A. Koster, and E. Koloto. Branch and tree decomposition techniques for discrete optimization. *Tutorials in Operation Research: INFORMSNew Orleans*, pages 1–29, 2005.

[71] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of ACM*, 21:549–568, 1974.

[72] N. Imani and Q. Gu. Connectivity is not a limit for kernelization: planar connected dominating set. In *the 9th Latin American Theoretical Informatics Symposium (Latin 2010) LNCS 6034*, pages 26–37, 2010.

[73] T. Jhansson and Carr-Motyckova. Reducing interference in ad hoc networks through topology control. In *the 2005 joint workshop on Foundations of mobile computing*, pages 17–23, 2005.

[74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Science*, 9(3):256–278, 1974.

[75] I.A. Kanj and L. Perkovic. Improved parameterized algorithms for planar dominating set. *In Proc. of the 27th MFCS. LNCS 2420*, pages 399–410, 2002.

[76] G. Karakostas. A better approximation ratio for the vertex cover. *ACM Transactions on Algorithms (TALG)*, 5(4):1–8, 2009.

[77] D. Kobler and U. Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126:197–221, 2003.

[78] S. Lee, Y. Ho, and R.M. Low. On the integer-magic spectra of maximal planar and maximal outerplanar graphs. *Congressus numberantium*, 168:83–90, 2004.

[79] X. Li. Algorithmic, geometric and graphs issues in wireless networks. *Journal of Wireless Communications and Mobile Computing (WCMC)*, 3(2):119–140, 2003.

[80] X. Li, P. Wan, and Y. Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *Tenth International Conference on Computer Communications and Networks*, pages 564–567, 2001.

[81] R.J. Lipton and R.E. Tarajan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[82] R.J. Lipton and R.E. Tarajan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[83] C. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1963.

[84] C. Liu and Y. Song. Exact algorithms for finding the minimum independent dominating set in graphs. *Lecture Notes in Computer Science*, 4288:439–448, 2007.

[85] H. Liu, P. Wan, C. Yi, S. Makki, and N. Pissinou. Maximal lifetime scheduling in sensor surveillance networks. *IEEE transactions on parallel and distributed systems*, 17(12):1526–1536, 2006.

[86] D. Lokshtanov, M. Mnich, and S. Saurabh. Linear kernel for planar connected dominating set. In *the 6th Annual Conference on Theory and Applications of Models of Computation*, pages 281–290, 2009.

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

[88] k. Mehlhorn and S. Naher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[89] R.H Mohring and F.J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Math*, 19:257–356, 1984.

[90] T. Nishizeki, N. Chiba, and N. Satio. An approximation algorithm for the maximum independent set problem on planar graphs. *SIAM Journal on Computing*, 11(4):663–675, 1982.

[91] R. Norman, F. Harary, and D. Cartwright. *Structural models: an introduction to the theory of directed graphs*. Wiley, 1966.

[92] I.H. Osman. *Meta-heuristics: theory and applications*. Springer, 1996.

[93] A.K. Parekh. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information Processing Letters*, 39:237–240, 1991.

[94] C.R. Reeves. *Modern heuristic techniques for combinatorial problems*. Halsted Press, 1993.

[95] G. Reinelt. Tspliba travelling salesman library. *ORSA Journal on Computing*, 3:376–384, 1991.

[96] N. Robertson and P.D. Seymour. Graph minors I: Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[97] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

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

[99] L.A. Sanchis. Experimental analysis of heuristic algorithms for the dominating set problem. *Algorithmica*, 33(1):3–18, 2002.

[100] C. Savage. Depth-first search and the vertex cover problem. *Information processing letters*, 14:233–235, 1982.

[101] G. Schaeffer. Random sampling of large planar maps and convex polyhedra. In *the 31st Annual ACM Symposium on the Theory of Computing,STOC99*, pages 760–769, 1999.

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

[103] Q.A. Shcherbina. Tree decomposition and discrete optimization problems: A survey. *Cybernetics and system analysis*, 43(4):549–562, 2007.

[104] H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. In *ESA2003*, pages 765–775, 2003.

[105] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[106] J. M. M. Van Rooji and H.L. Bodlaender. Design by measure and conquer: A faster exact algorithm for dominating set. In *the 25th Annual Symposium on theoretical aspects of computer science STACS 2008*, pages 657–668, 2008.

[107] J. M. M. Van Rooji and H.L. Bodlaender. Exact algorithms for edge domination. In *the 3rd international conference on Parameterized and exact computation*, volume 5018, pages 214–225. Spring Verlag, Lecture notes in computer science, 2008.

[108] V.V. Vazirani. *Approximation Algorithms*. Springer Verlag, Berlin, 2001.

[109] D.B. West. *Introduction to Graph Theory*. Prentice Hall Inc., 1996.

[110] M. Yannakakis and F. Garvil. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

[111] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of ACM*, 49:289–317, 2002.

Appendix A

Branchwidth of random maximal graphs

We show that the instances of Class (1) (the random maximal graphs and their subgraphs generated by LEDA) has branchwidth at most four. We prove this statement by constructing a branch-decomposition of width at most four for any maximal graph in this Class. Let G_n be a maximal graph of n vertices in Class (1). For $n = 3$, G_n is the graph with three edges (see Figure A.1). For $n \geq 4$, G_n is created by adding a new vertex in a randomly chosen face f of G_{n-1} and three edges between the new vertex and the three vertices incident to f [1, 88]. Let $E_3 = E(G_3)$. For $4 \leq j \leq n$, let u_j and E_j be the new vertex and the set of edges, respectively, added to create G_j. For each E_i, $3 \leq i \leq n$, we create a rooted binary tree T_i with root r_i and three leaves (see Figure A.1). We assign the three edges of E_i to the leaves of T_i, one edge per leaf in an arbitrary way. We say E_i is a parent of E_j if $i < j$ and the end vertices of the three edges of E_j except u_j are also end vertices of edges of E_i. If E_i is a parent of E_j, E_j is called a child of E_i. Obviously for $j \geq 4$, each E_j has a unique parent and for $i \geq 3$, each E_i has at most three children. We merge the rooted binary trees T_i , $3 \leq i \leq n$, into a rooted binary tree T by the following recursive procedure.

Merge-Tree(E_i)

1. *If E_i has any child E_j then call Merge-Tree(E_j) for every child E_j of E_i; otherwise output the rooted binary tree T_i and RETURN.*

2. *Merge the rooted trees obtained from Merge-Tree(E_j) for all E_j by connecting the roots of the binary trees into one rooted binary tree T; merge T and T_i by connecting the*

roots of them into a rooted binary tree T (see Figure A.1 for the merge process); output T and RETURN.

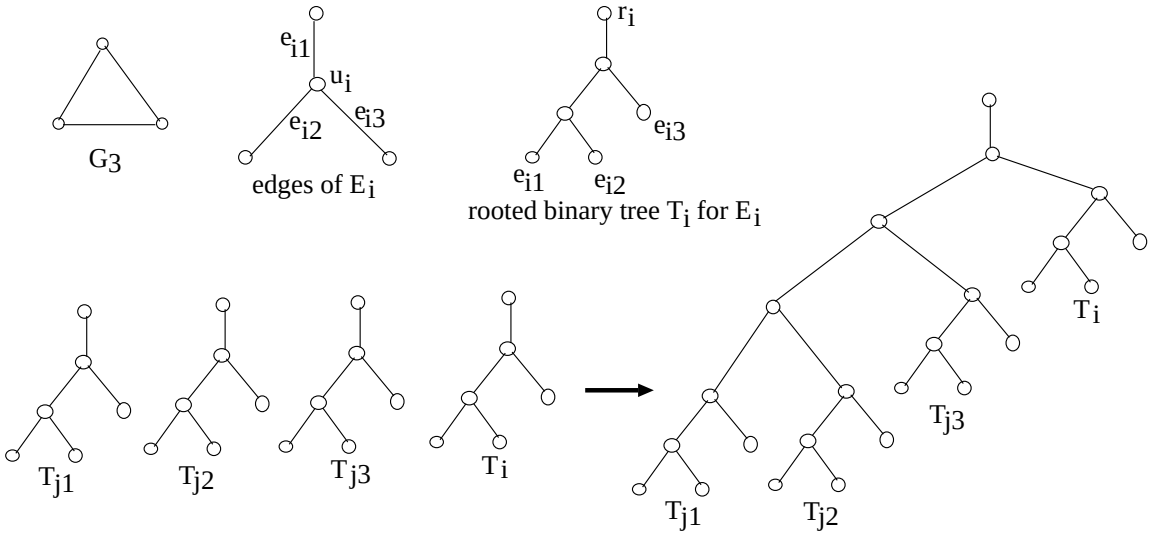


Figure A.1: Merge the rooted binary trees T_i into one binary tree.

Calling *Mereg-Tree(E_3)* merges all rooted binary trees T_i, $3 \leq i \leq n$, into a rooted binary tree T. For an arbitrary link e of T, let T_e be the subtree consisting of all descendant links of e in T. For a link e of T such that T_e has at most two leaves, $|S_e| \leq 4$. For a link e such that T_e has at least three leaves, T_e has at least one rooted binary tree T_j as a subtree. For such a link e, an E_j is maximal in T_e if T_e does not have a subtree T_i for E_i such that E_i is a parent of E_j. If T_e has only one maximal E_j then $Se \subseteq V(E_j)$, where $V(E_j)$ is the set of end vertices of edges in E_j. Assume that T_e has more than one maximal E_j. Let E_i be the parent of those E_js. Then $S_e \subseteq V(E_i)$. In either cases, $|S_e| \leq 4$. Therefore, T is a branch-decomposition of G_n (we need to remove the root of T) with width at most four. This implies that G_n has branchwidth at most four. It is known that the branchwidth of a subgraph of G_n is at most the branchwidth of G_n. Thus, the instances of Class (1) have branchwidth at most four.

Appendix B

The pre-processing for fast matrix multiplication

Applying dynamic programming is the most time consuming part of DPBF Algorithm and improving it is an interesting challenge. In dynamic programming step, we calculate the partial solutions for $H[A_e]$ for every internal link e with children e_1 and e_2, from the colorings of $\partial(A_{e_1})$ and those of $\partial(A_{e_2})$. As we described in Chapter 5, in the index method we put all partial solutions of $H[A_{e_1}]$ in a table T_1 with 6^{b_1} entries and similarly partial solutions of $H[A_{e_2}]$ in a table T_2 with 6^{b_2} entries. The entries of T_1 and T_2 are indexed by $1, 2, ..., 6^{b_1}$ and $1, 2, ..., 6^{b_2}$, respectively. In order to store the partial solutions of $H[A_e]$, we create a table T with 6^b entries. Using matching groups of partial solutions of $H[A_{e_1}]$ and $H[A_{e_2}]$ we calculate the basic color of partial solutions of $H[A_e]$. For each coloring $\eta_1 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_1}$ in a group S_1 of T_1, we choose every coloring $\eta_2 \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{b_2}$ in every matching group S_2, compute the disjoint components of $H[D_{e_1}(\eta_1) \cup D_{e_1}(\eta_2)]$, find the coloring $\eta \in \{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^b$ from η_1 and η_2, and calculate $a_e(\eta)$ as $a_e(\eta) = a_{e_1}(\eta_1) + a_{e_2}(\eta_2) - (X_3 \cup X_4)_{\#1}$, where $(X_3 \cup X_4)_{\#1}$ is the number of vertices in $X_3 \cup X_4$ which are colored by 1. For every η computed from η_1 and η_2 we store the minimum $a_e(\eta)$ and the corresponding $D_e(\eta)$ which is computed from $D_{e_1}(\eta_1) \cup D_{e_2}(\eta_2)$.

Dorn in [40] gives a new technique for combining dynamic programming and matrix multiplication to improve the speed of solutions for NP-complete problems such as PLANAR DOMINATING SET and PLANAR HAMILTONIAN CYCLES. This method is based on the distance product of matrices. The distance product of two $(n \times n)$-matrices A and B,

denoted by $A \star B$, is a $(n \times n)$-matrix C such that

$$c_{ij} = min_{1 \leq k \leq n}\{a_{ik} + b_{kj}\}, 1 \leq i, j \leq n \tag{B.1}$$

If the conventional matrix multiplication is applied the distance product of two $(n \times n)$-matrices takes $O(n^3)$ time. Using the fast matrix multiplication with exponent ω, the distance product of two $(n \times n)$-matrices whose elements are taken from $\{-m, ..., 0, ..., m\}$ can be computed in $O(m.n^{\omega})$. In addition, the distance product of two $(n \times p)$- and $(p \times n)$-matrices with $p > n$ can be computed in $O(p.(m.n^{\omega-1}))$ [111].

The main idea of combining dynamic programming and distance product is to arrange the partial solutions of the children e_1 in e_2 in matrices A and B such that $A \star B$ contains the partial solutions for e. Dorn in [40] lists a set of NP-complete problems whose time complexities can be improved using distance product and fast matrix multiplication. Combining dynamic programming and fast matrix multiplication is straightforward for some of these problems such as VERTEX COVER, INDEPENDENT SET and DOMINATING SET problems. He also stated that this method is not "immediately clear" for PLANAR HAMILTONIAN CYCLE and PLANAR LONGEST PATH problems. The common property of these problems is that in dynamic programming step after combining partial solutions of the children e_1 and e_2 of an internal link e, some post-processing must be applied to uncover forbidden solutions and calculate the colors of vertices in $\partial(A_e) = X_1 \cup X_2 \cup X_3$. The suggested idea to combine dynamic programming and fast matrix multiplication for this kind of problems is to replace the post-processing step with some pre-processings steps and change the entries of the child matrices based on the change coloring of the parent. The PLANAR CDS problem also needs some post-processing to identify forbidding components and choose the color of black vertices of $\partial(A_e)$ from $\{1_[, 1_], 1^*, \hat{1}\}$ according to the resulting disjoint components. In what follows, we describe how we can combine distance product and dynamic programming for the DBPF Algorithm.

B.1 Distance product and DPBF Algorithm

In this section we show that applying distance product in dynamic programming step of DPBF Algorithm can improve its time complexity. However, this result is not practical since the existing fast matrix multiplication methods are not efficient practically.

As we mentioned in Chapter 5, the goal of pre-processing step is to decide the colors

of black vertices in $X_1 \cup X_2 \cup X_3$ form $\{1_[, 1_], 1^*, \hat{1}\}$ only based on the colors X_4 vertices in both children. This pre-processing can also identify the forbidding components. The running time of this step is $O(|X_4|) = O(bw)$, the overall time complexity of the DPBF Algorithm will not be increased.

The main challenge in applying distance product for the DPBF Algorithm is how to define the entries of child matrices such that applying distance product generates the partial solutions for the parent link. We solve this challenge as follows. Let e be an internal link with two children e_1 and e_2 and let $|X_i|$ be the size of X_i for $1 \leq i \leq 4$. For every coloring $\{0, \hat{0}, 1_[, 1^*, 1_], \hat{1}\}^{|X_3|}$ of X_3 we define three tables A, B and C corresponding to e_1, e_2 and e. In Chapter 5 we show that there are at most $Z \leq 16^{|X_4|}$ matching for X_4. Two colors η_1 and η_2 are matched, if for every vertex $u \in X_4 : u$ has a basic color 1 in η_1 and η_2, or $\eta_1(u) = 0$ and $\eta_2 = \hat{0}$, or $\eta_1(u) = \hat{0}$ and $\eta_2 = 0$

For every coloring η_3 of X_3 vertices we define matrices A, B and C. for simplicity we assume that $X_3 = \emptyset$. For every coloring η we identify all matching pairs η_1 and η_2 which form η and put the partial solutions of η_1 to a specific row of A and the partial solutions of η_2 to a specific column of B. We define A as a matrix with $6^{|X_1|}$ and Z columns for the partial solutions by η_1. Similarly we define B as a matrix with Z rows and $6^{|X_2|}$ columns for the partial solutions by η_2. Each row of A is indexed by a color of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{|X_1|}$ and each column of B is indexed by a color of $\{0, \hat{0}, 1_[, 1_], 1^*, \hat{1}\}^{|X_2|}$. For a coloring η, assume that the ith row of A corresponds to the colors of vertices in X_1 from η and the jth column corresponds to the colors of vertices in X_2 from η. Assume that there are q pairs of η_1 and η_2 which form η. For each pair of η_1 and η_2, we put the partial solution by η_1 $(a_{e_1}(\eta_1) - (X_4)_{\#1})$ to an element in the ith row of A and the partial solution by η_2 (a_{e_2}) to an element in the jth column of B such that $(A[i,k], B[k,j])$, $1 \leq k \leq q$, correspond to the q pairs of η_1 and η_2. Then $C[i,j] = \min_{k=1}^{q}\{A[i,k] + B[k,j]\}$ gives the partial solution by η.

For every matching colors η_1 and η_2, the goal of pre-processing step is to find the corresponding row of A and column of B, to put the partial solutions of η_1 and η_2, only based on the color of vertices of X_4 in η_1 and η_2. Using pre-processing, the running time of DPBF algorithm is the same as the running of the distance product. Calculating C takes time $O(6^{(\omega-1)|X_1|}.6^{|X_3|}.Z^{|X_4|})$ in the worst case (i.e. when $|X_1| = |X_2|$). If we maximize this running time with the following constraints: $|X_1| + |X_2| + |X_3| + |X_4| \leq 1.5bw$, $|X_1| + |X_3| + |X_4| \leq bw$, $|X_1| + |X_2| + |X_3| \leq bw$ and $|X_2| + |X_3| + |X_4| \leq bw$, the running time will be $O(6^{(\omega-1)bw/2}.Z^{bw/2})$. With $\omega = 2.376$, the running time of DBPF Algorithm is improved to

$O(2^{8.08\sqrt{n}})$. The result of this appendix can be summarize in the following theorem.

Theorem B.1.1 *DPBF Algorithm solves the CDS problem for a plane graph G of n vertices in $O(2^{8.08\sqrt{n}}\gamma_c(G) + n^3)$.*

B.2 The pre-processing for fast matrix multiplication

Here we describe how the colors of vertices of $X_1 \cup X_2$ in η can be computed based on the colors of vertices in X_4 from η_1 and η_2. In what follows we define pre-processing. The pre-processing has three phases.

- **Phase1:** collecting necessary information on disjoint components in A_{e_1} and A_{e_2} with some vertices in X_4.
- **Phase2:** computing all possible matchings for black vertices in X_4, and collecting information about the disjoint components which are connected with that matching.
- **Phase3:** computing the colors of vertices in $X_1 \cup X_2$ based on the information of Phase 1 and Phase 2.

The first two phases are applied once at the beginning of the DPBF Algorithm. However the last phase is applied for every internal link of sphere-cut decomposition. Let T be an optimal sphere-cut decomposition of the input graph G with branchwidth b. Let e be an internal link of T with children e_1 and e_2. Colorings η_1 and η_2 are defined on A_{e_1} and A_{e_2}, respectively. We assumed that the colorings η_1 and η_2 be ordered in clockwise and counter-clockwise order from the first vertex of X_4, respectively. In what follows we describe three phases of pre-processing in details.

B.2.1 Phase 1

In this phase all possible colorings for vertices in X_1 and X_2 are computed with this respect that these colorings are part of colorings η_1 and η_2 we call these colorings *partial colorings*. For example, let η_1 is $\{1_[, 1*, 1*, \hat{1}, 1_[, 1*, 1_], 1_]\}$ which is defined on vertices $\partial(A_{e_1}) = \{u_1, u_2, ..., u_8\}$ and for $X_1 = u_6, u_7, u_8$, the partial coloring $\eta' = \{1*, 1_], 1_]\}$. For every partial coloring, η', we also keep some information about the disjoint connected components η'. In what follows we just describe the phase for X_1 and η_1, the phase is the same for X_2 and η_2.

There are at most 6^{bw} partial colorings, $\{0, \hat{0}, 1_[, 1^*, 1_], \hat{1}\}^{bw}$, for X_1. For every possible partial coloring, η', by checking the colors of vertices, we can identify the disjoint connected components. As we described in Chapter 5, there are two types of components: some components with only one vertex on η_1 , or some components with more than one vertices in η_1. For the components with more than one vertex in η_1, there is a small end, with color $1_[$, large end, with color $1_]$ and middle vertices, with color 1^*. Since every possible coloring, η' is a part of complete coloring η_1, another disjoint component can be appear in η'. These disjoint components are some components with small end in X_4 and some vertices in X_1, we call these disjoint components *open components*. For every partial coloring η' we keep the ordered list of open components. For example η' in the above example has two open components,$P'_1 = \{u_6, u_7\}, P'_2 = \{u_8\}$. It is clear that this phase can be completed in $O(6^{bw}.bw)$.

B.2.2 Phase 2

In this phase we compute all possible matchings for vertices in X_4. For every matching M between η_1 and η_2 we define $c(M)$ and $p(M)$ as follows:

- $c(M)$ includes the color of vertices of X_4 with basic color one in η_1 and η_2. For example if X_4 has k black vertices, $u_1, u_2, ..., u_k$,
  $c(M) = \{(\eta_1(u_1), \eta_2(u_1)), (\eta_1(u_2), \eta_2(u_2))..., (\eta_1(u_k), \eta_2(u_k))\}$.

- $p(M)$ is an ordered list. Every element of $p(M)$ has two integer numbers called c_1 and c_2. $p(M)$ is a list of pairs such that the first integer in every pair is c_1 and the second integer is c_2. c_1 (c_2) shows that how many consecutive disjoint open components in A_{e_1} (A_{e_2}) are connected together through the matching M. An element with $c_1 = 0$ and $c_2 = 0$ indicates a forbidding component.

Computing all possible matchings is similar to the method to compute forbidding components in Chapter 5. We compute all possible matching by induction on the number of vertices in X_4 with basic color 1. Let m be the number of vertices with basic color 1 in X_4 in η_1 and η_2.

For $m = 1$ there are three possible matchings M_1, M_2 and M_3, as follows :

- M_1 with $c(M_1) = \{(1_[, 1_[)\}$ and $p(M_1) = \{(1, 1)\}$.

- M_2 with $c(M_2) = \{(1_[, 1)\}$ and $p(M_2) = \{(1, 0)\}$.

$(\eta_1(u), \eta_2(u))$	$p(M_i)$ modifications
$(1_[, 1_[)$	add a new element $(1, 1)$ at the end of $p(M_i)$
$(1_[, 1^*)$	Find last element of $p(M_i)$ with $c_2 > 0$ increase c_1 by one in that element
$(1_[, 1_])$	Find last element of $p(M_i)$ with $c_2 > 0$ increase c_1 by one and decrease c_2 by one in that element
$(1_[, 1)$	add a new element $(1, 0)$ at the end of $p(M_i)$
$(1^*, 1_[)$	Find last element of $p(M_i)$ with $c_1 > 0$ increase c_2 by one in that element
$(1^*, 1_])$	Find last element of $p(M_i)$ with $c_2 > 0$ decrease c_2 by one in that element
$(1_], 1_[)$	Find last element of $p(M_i)$ with $c_1 > 0$ increase c_2 by one and decrease c_1 by one in that element
$(1_], 1^*)$	Find last element of $p(M_i)$ with $c_2 > 0$ decrease c_1 by one in that element
$(1_], 1_])$	Find last element of $p(M_i)$ with $c_1 > 0$ or $c_2 > 0$ decrease c_2 by one, and decrease c_1 by one in that element
$(1_], 1)$	Find last element of $p(M_i)$ with $c_1 > 0$ decrease c_1 by one in that element
$(1, 1_[)$	add a new element $(0, 1)$ at the end of $p(M_i)$
$(1, 1_])$	Find last element of $p(M_i)$ with $c_2 > 0$ decrease c_2 by one in that element
Others	Nothing

Table B.1: The rules to compute $p(M_{i+1})$ from $p(M_i)$

- M_3 with $c(M_3) = \{(1, 1_[)\}$ and $p(M_3) = \{(0, 1)\}$.

As we mentioned in Chapter 5, to find all possible matchings with $i + 1$ black vertices from matching with i black vertices, we add a new black vertex to X_4 and we add one of the following 15 possible pairs of colors to $c(M_i)$ to generate $c(M_{i+1})$.

$$\begin{aligned}\{ \quad & (1_[, 1_[), (1_[, 1^*), (1_[, 1_]), (1_[, \hat{1}), (1_], 1_[), (1_], 1^*)(1_], 1_]), (1_], \hat{1}),\\ & (1^*, 1_[), (1^*, 1^*), (1^*, 1_]), (1^*, \hat{1}), (\hat{1}, 1_[), (\hat{1}, 1^*), (\hat{1}, 1_])\}.\end{aligned}$$

Table B.1 shows that how $p(M_i)$ is modified to generate $p(M_{i+1})$.

Using Table B.1 we compute all possible matchings for $m = 1, 2, ..., bw$. During computing a matching M if $p(M)$ contains $(0, 0)$ a forbidding component appears and if $p(M)$ contains some elements with negative values, the matching is not valid. The running time of this step is $O(16^{bw})$.

B.2.3 Phase 3

Let η_1 and η_2 are the colorings corresponding to the child links e_1 and e_2 in T. In $O(bw)$ time, we can identify partial colorings η_1' and η_2' and the corresponding matching M, for η_1 and η_2. Using the information of phase one, the list of open components in these partial colorings are also available. Using $p(M)$ we can identify the open components which are connected together and we can update the colors of vertices in the open components in $X_1 \cup X_2$. In this step we identify the color of $X_1 \cup X_2$ based om the matching (vertices of X_4) and in distance product we can identify the corresponding row i in A and column j in B to put the partial solutions of η_1 and η_2.

Appendix C

An upperbound for the size of CDS of a partially triangulated $(r \times r)$ -grid

Recall from Chapter 5, a partially triangulated $(r \times r)$-grid R is a graph obtained by adding edges between pairs of nonconsecutive vertices on a common face of a planar embedding of an $(r \times r)$-grid. In this appendix we prove that for a partially triangulated $(r \times r)$-grid graph R, the size of CDS is at most $\frac{r^2}{5} + o(r)$.

Let the rows and columns of R are numbered from 0 to $r-1$. We divide R into $m = \lceil \frac{r}{5} \rceil$ sunbgraphs, $R_1, R_2, ..., R_m$, such that every subgraph R_i, $1 \leq i \leq m-1$ is a subgraph of R induced by five rows $(i-1)5, (i-1)5+1, ..., (i-1)5+4$. The last subgraph R_m may have less than five rows. We redefine the general dominating set of R_i, introduced in Chapter 6, as a subset D_i of vertices of R_i such that it dominates all vertices in $U = \{(x,y) | x = (i-1)5+1, (i-1)5+2, (i-1)5+3, 1 \leq y \leq r-2\}$ (there are some non-dominated vertices on the first and last rows and columns of R_i). We call U the set of internal nodes of R_i. In what follows we first show that for every subgraph R_i there is a general dominating set, D_i, of size r. Next we show that the union of these general dominating sets and including $4r$ vertices of R creates a connected dominating set of R.

Let R_i, $1 \leq i \leq m$, be a subgraph of R from the definition in the previous paragraph that includes rows $x_0, x_1, ..., x_4$. We define a general dominating set of R_i, D_i, as follows:

$$D_i = \{(x_l, j) | 0 \leq l \leq 4, j \bmod 5 = l\} \tag{C.1}$$

$|D_i| = r$ since one vertex from every column is included in D_i and also D_i dominates all internal nodes of R_i. Figure C.1 shows D_i for a sugraph of a partially triangulated (20×20)-grid graph R.

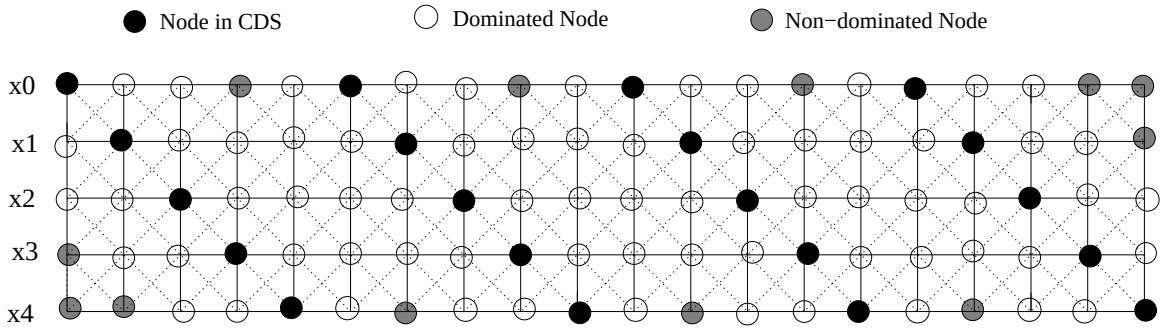


Figure C.1: A general dominating set of R_i (A subgraph of a partially triangulated 20×20-grid graph)

Let $D' = \bigcup_{1 \leq i \leq m} D_i$. It is clear that D' is a general dominating set of R with $U = \{(x, y) | 1 \leq x \leq r-2, 1 \leq y \leq r-2\}$ (See Figure C.2). Each connected component of D' has a vertex on row 0 or $r-1$. Including the vertices on these two rows, i.e. 0 and $r-1$, we generate a connected general dominating set of R, D'. If we include the vertices on columns 0 and $r-1$ to D', every vertex in R will be dominated. Thus, in summery:

$$D_c = \bigcup_{1 \leq i \leq m} D_i \cup \{(x, y) | x = 0, r-1 \text{ and } 0 \leq y \leq r-1\} \cup \{(x, y) | 0 \leq x \leq r-1 \text{ and } y = 0, r-1\} \tag{C.2}$$

D_c is a connected dominating set of R of size $\frac{r^2}{5} + 4r$.

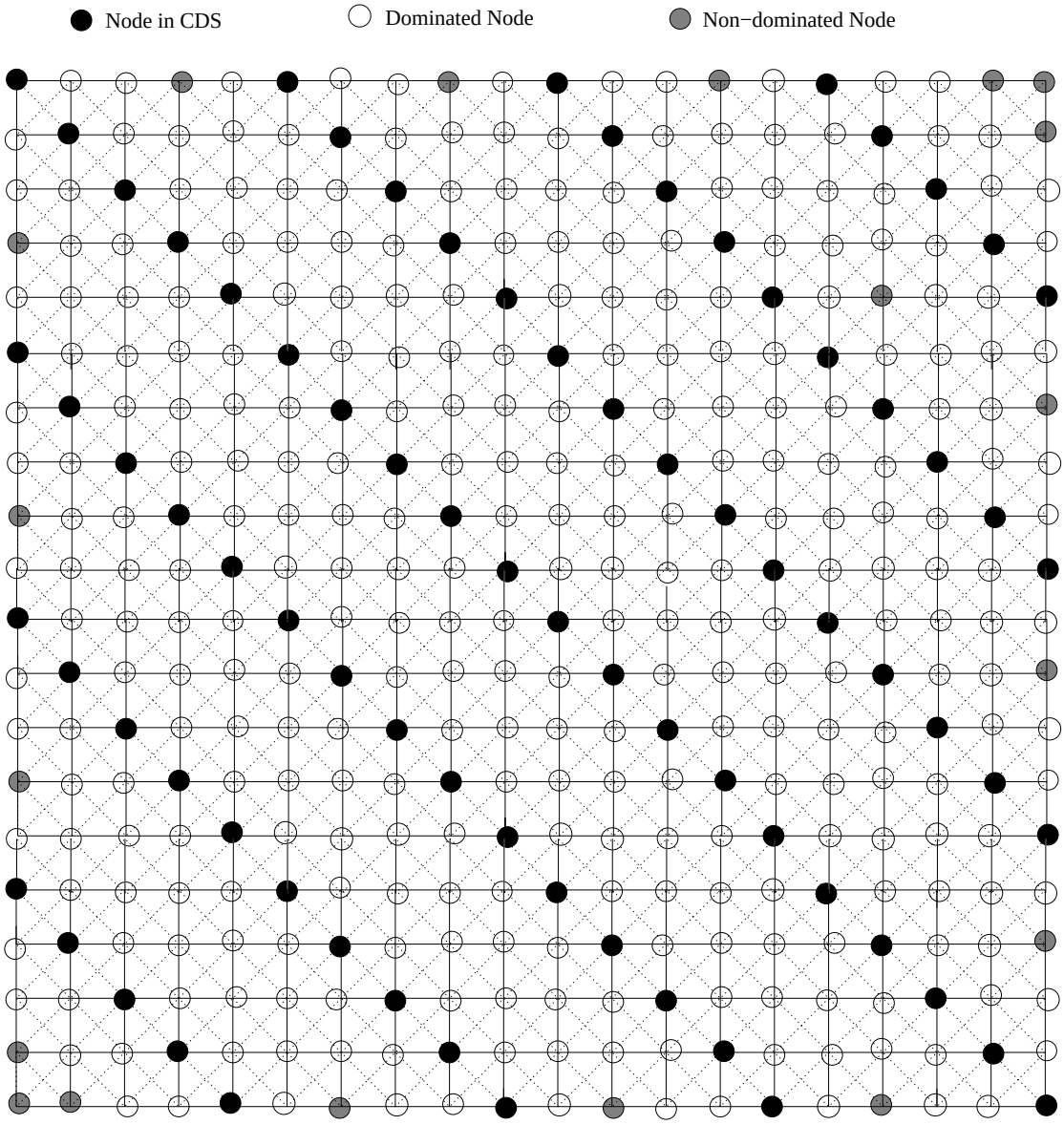


Figure C.2: D' is a general dominating set of a partially triangulated 20×20-grid graph