THE MAXIMUM STABLE SET PROBLEM IN PERFECT GRAPHS

by

Ahmet Çağrı Düzgün

B.S., Industrial Engineering, Boğaziçi University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2017

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my supervisor Assoc. Prof. Tınaz Ekim Aşıcı for her support and patience throughout my entire graduate study. I would also like to thank Assoc. Prof. Zeki Caner Taşkın for his invaluable contribution to this master thesis. This research would have been impossible without their guidance and encouragement.

I am very grateful to Prof. Temel Öncan for taking part in my thesis committee and for his valuable suggestions.

Furthermore, I thank TUBITAK for their financial support during my graduate study.

ABSTRACT

THE MAXIMUM STABLE SET PROBLEM IN PERFECT GRAPHS

Maximum Stable Set (MSS) problem is a well-known problem in graph theory. It is an NP-hard problem in general graphs and it has been extensively studied due to both its theoretical and practical importance. On the other hand, when MSS problem is limited to a class of graphs, called perfect graphs, it is polynomially solvable through a Semidefinite programming (SDP)-based algorithm. However, SDP solvers are known to be slow especially in larger problems, leading us to question the efficiency of the SDP-based algorithms. Moreover, there has been only limited number of studies examining the MSS problem solely in perfect graphs. Motivated by these, in this thesis, our objective is to compare performances of different algorithms with the SDP-based algorithm in perfect graphs. With this objective, we develop cutting plane algorithms that can solve MSS problem in perfect graphs. Additionally, investigate an existing branch-and-bound algorithm to see the performance of a powerful algorithm designed for MSS problem in general graphs. Then, we compare SDP-based algorithm with this branch-and-bound and our cutting plane algorithms using a number of perfect graph instances. Our comparison shows that the cutting plane algorithms perform considerably better than the other algorithms.

ÖZET

MÜKEMMEL ÇİZGELERDE EN BÜYÜK KARARLI KÜME PROBLEMİ

En büyük kararlı küme problemi çizgeler kuramındaki iyi bilinen bir problemdir. Bu problem genel bir çizge için NP-zor bir problemdir. Teorik ve uygulamadaki öneminden dolayı fazlaca çalışılmıştır. Problemi sadece çizge sınıflarının en önemli sınıflarından biri olan mükemmel çizgeler için ele aldığımız zaman, problemin polinom zamanda yarı kesin programlamaya dayanan bir algoritma ile çözüldüğünü görürüz. Fakat yarı kesin programlama problemlerinin özellikle fazla değişkenli olduğunda yavaş çözüldüğü de bilinir. Ayrıca, problemi sadece mükemmel çizgelerde inceleyen sınırlı sayıda yayın vardır. Tüm bunları göz önüne aldığımızda bizim amacımız yarı kesin programlamaya dayanan algoritma ile diğer algoritmaların performansını mükemmel çizgelerde karşılaştırmaktır. Bu amaçla biz kesen düzlem algoritmaları geliştirdik. Ayrıca, en büyük kararlı küme problemini herhangi bir çizgede çözen güçlü bir algoritma örneği görebilmek için güçlü bir dal-sınır algoritmasını inceledik ve sonra bu algoritmaları mükemmel çizge örnekleri kullanarak karşılaştırdık. Karşılaştırmalarımız kesen düzlem algoritmalarının diğerlerinden daha iyi çalıştığını ortaya çıkardı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	clique of graph G
$\overline{C}$	maximal clique of graph G
$\mathcal{C}$	set of all cliques of graph G
E	vertex set of graph G
$\|E\|$	cardinality of E
G	graph
$\overline{G}$	complement of graph G
$\mathcal{IC}$	set of initial cliques generated by the initial clique procedure
$N(v)$	neighborbood of vertex v in graph G
S	stable set of graph G
$\mathcal{C}$	set of all stable sets of graph G
V	vertex set of graph G
$\|V\|$	cardinality of V
$\mathcal{X}^S$	incidence vector of a stable set S
$\alpha(G)$	maximum cardinality of a stable set in graph G
$\theta(G)$	theta number of graph G
$\omega(G)$	maximum cardinality of a clique in graph G

1. INTRODUCTION

Graph theory studies graphs which reflects the pairwise relations between the objects through nodes and edges. Graphs can be used to model many problems in various fields such as computer science, operations research, communication networks, social and information systems. Along with that, graph theory also seeks answers to the problems arising from pure mathematical concepts. Considering both theoretical and practical importance of graph theory, it is not surprising that graph theory gave birth to various important problems which have been extensively studied for both theoretical and practical interest. In this thesis, we study one of these important problems, maximum stable set (MSS) problem, with the aim of finding the most efficient algorithm which can solve it in the one of the most important graph classes.

Given a graph $G = (V, E)$ with set of vertices V and set of edges E, a stable set of G is a set of vertices where any two vertices are non-adjacent and a clique of G is a set of vertices where any two vertices are adjacent. Stability number of G $\alpha(G)$ (MSS number) is defined as the maximum cardinality of a stable set in G, and clique number of G $\omega(G)$ is defined as maximum cardinality of a clique in G. A k-coloring of G is a vertex coloring in which labels of $1, ..., k$ are assigned to vertices of G provided that every vertex has a label and that no two adjacent vertices have the same label where k is integer and $k \geq 1$. The chromatic number of G, denoted by $\chi(G)$, is the smallest integer number k (or smallest number of colors needed) to color vertices of G. The complement of G is the graph with the same vertex set as G, but it has an edge between two vertices if and only if G does not have an edge between these two vertices. The complement of G is denoted by $\overline{G}$. In a graph, an odd hole is an induced subgraph which is a cycle of odd length at least five.

In graph theory, there is a large number of graph classes where a graph class is defined as the set of graphs sharing the same common property. Graphs classes enable development of efficient algorithms by exploiting common properties. One of the most important graph classes is perfect graphs. Perfect graphs are one of the graph

classes which has been extensively studied. A graph G is called perfect if the equality of $\omega(H) = \chi(H)$ holds for all induced subgraphs H of G. Perfect graphs were first introduced by Berge [2]. There are two important theorems about characterization of perfect graphs. The first theorem is called Weak Perfect Graph Theorem and was proved by Lovász [3]. It states that a graph is perfect if and only if its complement is perfect. The second one is called Strong Perfect Graph Theorem and was proved by Chudnovsky *et al.* [4]. It states that a graph is perfect if and only if itself and its complement do not contain an odd hole of length ≥ 5.

Maximum Stable Set (MSS) problem consists of finding a maximum stable set in a given graph G with corresponding vertices. Solving MSS in a general graph is NP-hard (See the proof in [5]). Since a clique (a complete subgraph) in a graph G corresponds to a stable set in $\overline{G}$, finding a maximum clique in G is equivalent to finding a stable set in $\overline{G}$. Therefore, these two problems are equivalent.

After giving some necessary definitions, we can now briefly explain the objective and the motivation of this thesis. While solving MSS provlem in general graphs is NP-hard, it can be solved in perfect graphs in polynomial-time by a semidefinite programming (SDP) algorithm [6]. However, performance of the SDP algorithm is not clear since being a polynomial-time algorithm does not necessarily imply being an efficient algorithm in practice. There are studies comparing performances of different methods employed for this problem in general graphs (See, for instance, [7] for an extensive review). However, to the best of our knowledge, there is no study in the literature comparing performances of different methods just for perfect graphs. Hence, we aim to compare the SDP algorithm with a cutting plane, and a branch-and-bound algorithm only on perfect perfect graphs.

2. LITERATURE REVIEW

MSS problem is one of the famous optimization problems which has been studied for a long time. It has wide range of applications spanning various areas such as stock market and financial networks [8], covering-location problems [9] and wireless networks [10]. We refer the reader to [11] for a detailed list of application areas.

Finding a MSS is NP-hard problem on general graphs and consequently, difficult to solve. Given practical and theoretical importance, and difficulty of the problem, a considerable effort has been devoted and many approaches were developed to tackle this problem. We here mention only main of approaches to give a brief idea.

Branch-and-bound approaches are the most popular methods applied by numerous studies for this problem. Hence, it is also not surprising that most recent and efficient approaches for MSS problem belongs to the class of branch-and-bound approaches such as [12–15]. In addition, different variants of branch-and-bound algorithms can be found in [16–19]. Branch-and-bound methods employ different strategies in both determining upper, lower bounds and branching, which creates the difference among their performances. Branch-and-cut approaches have been also proposed for MSS problem but not as many as branch-and-bound approaches. For an overview of effective branch and cut algorithms for this problem, we refer the reader to [20]. Other approaches include branch-and-price [21], partially enumerative algorithm [22], indefinite quadratic programming [23] and column generation [24]. An extensive review of MSS problem and list of both exact and heuristic approaches for this problem can be found in surveys [25] and [11]. Bomze *et al.* [25] provides insights into approaches it covered and describes some of the application areas of MSS problem such as coding theory and pattern recognition. However, it covers only the approaches developed before 1999 and there are many important approaches developed after that date. For an updated review of the MSS algorithms (especially for the ones developed after 1999), we refer the reader to Wu and Hao [11] which presents a computational comparison of ten approaches in terms of their performances using benchmark instances.

Giandomenico *et al.* [26] proposes an interesting approach which employs a cutting plane algorithm and uses SDP to obtain upper bound. They base their approach on the following. While LP relaxations take reasonably less time and provide weak upper bounds, SDP relaxations take longer time but provide much stronger upper bounds. Hence, they claim a branch-and-bound algorithm based on either of the relaxations are slow. They support their idea by saying that sophisticated mathematical programming algorithms for MSS have not been significantly more efficient than the relatively simple algorithms based on implicit enumeration. However, this idea is not valid for algorithms designed solely for perfect graphs since LP relaxations and SDP relaxations used for general graphs yields exactly the MSS number for perfect graphs.

When we limit ourselves only to polynomial algorithms, exact algorithms for MSS problem only on particular classes of graphs can be found in literature. In several classes of perfect graphs such as chordal, permutation and comparability graphs, MSS problem can be solved in polynomial-time with different approaches (See [27] for explanations). t-perfect graphs is another class of the perfect graphs where MSS problem can be solved in polynomial-time by Eisenbrand *et al.* [28]. In fact, a polynomial SDP-based algorithm also exists to solve MSS problem in perfect graphs due to Grötschel *et al.* [6]. This SDP-based approach is one of the main parts of our research and we examine it in detail in the following sections. Other than perfect graphs, in claw-free graphs, MSS problem can be solved in polynomial-time by Minty [29]. See [30] and [31] for polynomial-time algorithms in other classes of graphs.

Despite extensive research on MSS problem in general graphs, only few studies tackle MSS problem restricted to perfect graphs. Grötschel *et al.* [6] suggests a polynomial-time approach to extract a maximum stable set in perfect graphs by computing $\alpha(G)$ through SDP. Yıldırım and Fan-Orzechowski [1] also takes a similar but more practical and efficient approach by exploiting the properties of SDP solutions. To the best of our knowledge, it is the most efficient polynomial-time algorithm for MSS problem in perfect graphs. Both of these algorithms are discussed in the following sections. There are no other approaches designed specifically for perfect graphs we know of in the literature.

SDP algorithms are known to be slow in practice especially in large instances (See [32] and [33]). Hence, it is very likely that other exact algorithms for this problem can perform better than the SDP-based polynomial-time algorithm of Yıldırım and Fan-Orzechowski [1]. Motivated by this, our primary focus lies in comparing this polynomial-time algorithm in Yıldırım and Fan-Orzechowski [1] with a cutting plane algorithm we develop to tackle MSS problem in perfect graphs. Cutting plane algorithm is also different from the ones in the literature since perfect graphs have distinct properties general graphs does not have (See Chapter 3 for these properties). In literature, there is no study comparing this polynomial-time algorithm with other exact approaches. Hence, we aim to compare different algorithms that can solve MSS problem in perfect graphs.

In the next chapter, we present integer programming formulation of MSS problem for general graphs. We also present an LP relaxation for IP formulation of the MSS problem. LP relaxation provides an upper bound for the cardinality of MSS for a general graph. We then present a property of perfect graphs which implies that LP relaxation solves MSS problem exactly in perfect graphs. Based on this relaxation, we develop a cutting plane algorithm in Chapter 4. The SDP formulation of Grötschel [6], presented in Chapter 5, also provides an upper bound for the cardinality of MSS in general graphs while it finds the exact cardinality for perfect graphs. The SDP-based algorithm proposed by Yıldırım and Fan-Orzechowski [1] is discussed in Chapter 6 where we also introduce slightly different SDP-based algorithms. In Chapter 7, we compare the cutting plane algorithm and the SDP-based algorithm with an existing branch-and-bound algorithm from the literature by computationally testing them on perfect graph instances.

3. IP FORMULATION

Given a graph $G = (V, E)$, the maximum stable set (MSS) problem in G can be easily formulated as a 0-1 integer programming (IP) formulation, which we call BasicIP, as follows:

$$max \quad \sum_{i \in V} x_i \tag{3.1}$$

$$s.t. \quad x_i + x_j \leq 1 \quad \forall (i,j) \in E \tag{3.2}$$

$$x_i \in \{0,1\} \quad \forall i \in V \tag{3.3}$$

where variable x_i is 1 if and only if vertex i is in the stable set and 0 otherwise.

BasicIP model can be strengthened by adding valid inequalities. One of the most important valid inequalities for MSS problem is clique inequalities [34]. They are clearly valid inequalities since only one of the vertices of a clique can be in the stable set. They can be given as follows:

$$\sum_{i \in C} x_i \leq 1 \quad \forall C \in \mathcal{C} \tag{3.4}$$

where $\mathcal{C}$ the represents the set of all cliques in G. In a graph, a maximal clique is a clique which cannot be extended by adding another vertex of the graph. If we generate clique inequalities for all maximal cliques, then we don't need to generate any other clique inequalities. This is simply because maximal cliques dominate all the others. Hence, we can introduce a new IP model to solve MSS problem, which we call MCliqueIP:

$$max \quad \sum_{i \in V} x_i \tag{3.5}$$

$$s.t. \sum_{i \in \overline{C}} x_i \leq 1 \quad \forall \overline{C} \in \mathcal{C} \tag{3.6}$$

$$x_i \in \{0,1\} \quad \forall i \in V \tag{3.7}$$

where $\overline{C}$ is a maximal clique. We now define two polytopes which prove to be useful in developing an LP model to solve MSS problem in perfect graphs. We first define incidence vector before defining polytopes.

Incidence vector $\mathcal{X}^S$ of a stable set $S \subseteq V$ is defined by

$$\mathcal{X}_i^S = 1 \text{ if } i \in S,\ \mathcal{X}_i^S = 0 \text{ if } i \notin S \tag{3.8}$$

$STAB(G)$ is a polytope denoting the convex hull of incidence vectors of all stable sets in G and is defined as follows:

$$STAB(G) = conv\{\mathcal{X}^S \in \Re^{|V|} | S \subseteq V \text{ stable set}\} \tag{3.9}$$

Another polytope, called fractional stable set polytope of G, is defined as follows:

$$QSTAB(G) = \{x \in \Re_+^{|V|} : \text{satisfying} \quad (3.4)\}$$

where x is a vector of size $|V|$.

It is easy to see that $QSTAB(G)$ contains STAB(G) since extereme points of $STAB(G)$ are also contained in $QSTAB(G)$. We present a very important property of perfect graphs which leads to construction of an LP model. In perfect graphs, the following holds due to Grötschel *et al.* [35]:

$$STAB(G) \subseteq QSTAB(G) \tag{3.10}$$

Hence, we can find a maximum stable set in a perfect graph by solving the following LP model, which we call CliqueLP:

$$max \quad \sum_{i \in V} x_i \tag{3.11}$$

$$s.t. \sum_{i \in C} x_i \leq 1 \quad \forall C \in \mathcal{C} \tag{3.12}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{3.13}$$

We now present a theorem demonstrating importance of maximal clique inequalities to solve MSS problem by linear programming. Before giving the theorem, we should first define facet-defining inequalities. The *facet-defining inequalities* are valid inequalities with the maximum possible dimension which is one less than the dimension of a polyhedron. All of the other valid inequalities are dominated by facet-defining inequalities and facet-defining inequalities are adequate to completely describe the polyhedron.

Theorem 3.1. *[36] Let G be a graph with node set V and $\mathrm{C} \subseteq V$ be a clique. An inequality $\sum_{i \in \mathrm{C}} x_i \leq 1$ is a facet of STAB(G) if and only if C is a maximal clique in G.*

Theorem 3.1 and Equality (3.10) show maximal clique constraints, along with nonnegativity constraints, suffice to define $STAB(G)$ of a perfect graph. Therefore, we can solve MSS problem in perfect graphs with the following LP model, which we call MCliqueLP:

$$max \quad \sum_{i \in V} x_i \tag{3.14}$$

$$s.t. \sum_{i \in \overline{C}} x_i \leq 1 \quad \overline{C} \in \mathcal{C} \tag{3.15}$$

$$x_i \geq 0 \quad \forall i \in V \tag{3.16}$$

It is a well-known fact that linear programming problems can be solved in polynomial-time [37]. However, this does not mean that every LP problem is polynomially solvable since polynomiality of an LP problem also depends on size of the input. The following theorem shows that MCliqueLP problem cannot be solved in polynomial-time.

Theorem 3.2. *[38] The optimization problem for QSTAB(G) is NP-hard in general.*

4. CUTTING PLANE ALGORITHM

Given a perfect graph G, a maximum stable set in G can be found by solving MCliqueLP. However, it is not efficient to generate all the maximal cliques of a graphs to find a maximum stable set. Instead, we implement a cutting plane algorithm. The cutting plane algorithm tries to find an optimal solution by explicitly considering fewer number of constraints. It constructs an initial LP with the same objective as the original problem but with a subset of original constraints. Initial maximal clique (constraint) generation can be very important for efficiency of a cutting plane algorithm. Hence, we experiment with several heuristics to generate the initial constraints. After the generation of initial constraints, the algorithm runs iteratively and each iteration consists of two steps. Basically, in the first step, the main problem is solved over the maximal cliques constraints and resulting x_i values are given as parameters to the subproblem. The LP in the first step can be seen as an approximation of the original problem. In the second step, the subproblem (a separation problem) is solved to add another original constraint to the LP in the first step in the hope of improving the approximation. Given the solution of main problem, the separation problem aims to find a most violated maximal clique.

In the first step, current LP is solved to optimality. If the solution vector is integral (binary) and all edge constraints are satisfied, algorithm stops since the solution is optimal. That is because none of the constraints (3.15) can be violated by the current solution. Therefore, it lies in the feasible region defined by inequailities (3.15). If the solution is not integral, then we proceed to the second step in which cutting plane(s) are generated and added to the LP. This first and second step is repeated iteratively until an integral solution is found. Cutting planes are inequalities violated by the current optimal solution of step 1 but they are valid inequalities for the polyhedron defined by inequalities (3.15). Hence, LP is strengthened by adding these violated inequalities. The procedure of identifying violated valid inequalities is called *separation*. Even though a separation procedure generally tries to find a most violated inequality, it may be better to identify more than one violated inequalities to compute the solution.

Facet-defining inequalities are preferred as violated valid inequalities since they are the only inequalities to completely define the polyhedron of the feasible region. Since maximal clique constraints are facet-defining in perfect graphs, we focus solely on maximal clique constraints. We don't include other type of valid inequalities such as rank inequalities employed in Branch and Cut algorithm in literature (See [39] for example). Hence, only violated maximal cliques are identified in the second step. After solving the LP in the first step, the algorithm checks all edges and 3-cycles to see if there is an edge or a 3-cycle violated. If it detects one, a maximal clique containing edge or 3-cycle is generated and added to the LP. Then, LP is resolved. Algorithm continues to the step 2 only if there exists no edge or 3-cycle inequality violated.

Identification of a most violated maximal clique is, in fact, a weighted maximum clique problem where weight of a node corresponds to the value of that node in the current solution. Since weighted maximum clique problem is also NP-hard [40], it cannot be solved in polynomial-time. Therefore, separation algorithm also has a significant effect on the overall efficiency of the cutting plane algorithm. We formulate separation problem as an IP problem and solve it. We exploit the following proposition to reduce the size of the separation problem.

Proposition 4.1. *Given a graph $G = (V, E)$, let x^* be the optimal solution vector of the main problem. Define two sets of vertices P and U such that $P := \{i \in V | x_i^* = 0$ and $U := \{i \in V | x_i^* = 1\}$. The vertices in sets P and U do not need to be considered for the separation problem.*

Proof. Any vertex in P does not add to the total value of a clique in the weighted maximum clique problem. Hence, there has to be an optimal solution which does not contain them. Therefore, they can be removed from the problem. Any vertex in U cannot be contained in any violated clique. Since edge inequalities are already dominated by initial constraints in the main problem, there cannot be a violated edge inequality. It follows that a vertex i adjacent to a vertex in U must satisfy that $x_i^* = 0$. Assume a clique C exists such that C contains a vertex $j \in U$. Then, $x_i^* = 0$ for all $i \in U - \{j\}$. Therefore, $\sum_{i \in C} x_i^* = 1$ and this implies C cannot be a violated clique of

the main problem. □

By Proposition 4.1, we can suggest that vertices contained in either U or P can be removed once the main problem solved. They are not needed to be able to find the most violated cliques. Hence, we can construct the subproblem using only the vertices such that $x_i^* \neq 0$ or 1. However, it is not guaranteed that the subproblem gives a maximal clique though it gives certainly a most violated clique. That is because of the vertices in P. Hence, we have to consider the vertices in P one by one to make the resulting most violated clique maximal.

Given a graph $G = (V, E)$, main problem of the cutting plane algorithm is formulated as follows:

$$max \quad \sum\nolimits_{i \in V} x_i \tag{4.1}$$

$$s.t. \sum\nolimits_{v \in \overline{C}} x_v \leq 1 \quad \overline{C} \in \mathcal{IC} \tag{4.2}$$

$$x_i \geq 0 \quad \forall i \in V \tag{4.3}$$

where $\mathcal{IC}$ is the set of initial cliques generated by the initial clique procedure.

To introduce the subproblem, we need the optimal solution of the main problem since weights of the objective function of the subproblem are determined by this optimal solution vector. Denote the optimal solution of the main problem by x^*. Additionally, we define a new set of vertices R such that $R := \{i \in V | x_i \neq 0 \text{ or } 1\}$. R represents the remaining vertices after the removal of the unnecessary vertices according to Proposition 4.1. The subproblem is formulated as follows:

$$max \quad \sum\nolimits_{i \in R} x_i^* y_i \tag{4.4}$$

$$s.t.\ y_i + y_j \leq 1 \quad \text{for all } (i,j) \text{ such that } i, j \in R \text{ and } (i,j) \notin E \tag{4.5}$$

$$y_i \in \{0, 1\} \tag{4.6}$$

Additionally, before implementing these separation approaches, we may check all 3-cycles (three nodes that are adjacent to each other) to see whether they are violated. If a violated 3-cycle can be found, then it is lifted (made maximal) and added to the main problem as a clique constraint. We then directly go to the first step.

We introduce different initial cliques generation procedures knowing that they might generate different maximal cliques initially. Therefore, they have an effect on the performance of the cutting algorithm. We can learn whether one of these procedures performs better than the others in practice by running experiments. Hence, depending on different initial clique generation procedures, we experiment with different versions of the cutting plane algorithm. We present different clique generation procedures implemented in the following subsection. These different versions of the cutting plane algorithm have the same structure and show only minor differences. The general structure of the cutting plane algorithm is given in Figure 4.

4.1. Initial Clique Generation

As for the initial clique generation, there are different approaches we implement. As a first approach, a greedy clique cover heuristic is applied and then, resulting cliques are adjusted to be maximal cliques. We call this approach as ICG1. Pseudocode of ICG1 is given in Figure 4.1.

In the procedure ICG1, cliques constructed by greedy clique covering are made maximal by the greedy algorithm MakeMaximal which checks all vertices one by one in a random order. A vertex is added to the current clique if it is adjacent to all vertices in the clique at that moment. This procedure is presented in Figure 4.3.

Secondly, maximal initial cliques that contain every node at least once can be randomly generated as given in Figure 4.4 (procedure ICG2). Since all generated cliques are already maximal, we do not run the procedure MakeMaximal here.

INPUT :A perfect graph $G = (V, E)$ with set of vertices V and set of edges E

Generate all 3-cycles and store them in T

Generate initial cliques

Construct the main problem

repeat

Solve the main problem (4.1) and store the solution vector x^*

while There is violated edge or 3-cycle constraint **do**

for $(i, j, k) \in T$ **do**

if $x_i^* + x_j^* + x_k^* > 1$ **then**

Define a new set of vertices $K = \{i, j, k\}$

Apply MakeMaximal procedure to K.

Add the clique constraint of K to main problem.

end if

end for

for $(i, j) \in E$ **do**

if $x_i^* + x_j^* > 1$ **then**

Define a new set of vertices $K = \{i, j\}$

Apply MakeMaximal procedure to K.

Add the clique constraint of K to main problem.

end if

end for

Solve the main problem (4.1) and store the solution vector x^*

end while

Construct the subproblem using x^* and store the set $U := \{i \in V | x_i^* = 0\}$

Solve the subproblem (4.4) and store the solution vector y^*

Extract the violated clique from y^* and store the clique in C

for $i \in U$ **do**

if $(i, j) \in E \forall j \in U$ **then**

$C := C \cup \{i\}$

end if

end for

until x^* is an integral vector

Extact the MSS from x^*

Figure 4.1. Cutting Plane Algorithm.

INPUT : A graph with number of vertices n

Order the vertices and color first vertex with first color.

for vertex $i = 2$ to n **do**

For the current vertex i, find the lowest numbered color such that all colored vertices with this color are adjacent to the current vertex i. Then color vertex i with this color. If such a color does not exist, assign a new color to the vertex i by incrementing maximum color number by one.

end for¡

for all color set of vertices **do**

Apply procedure "MakeMaximal" to the current set of vertices

Generate clique inequality from the current set of vertices

end for

Figure 4.2. ICG1.

Neither ICG1 nor ICG2 ensure that edge inequalities are not violated. Hence, for these first two approaches, every new LP solution of first step is checked to see if a violated edge inequality can be found. If a violated one can be found, a maximal clique covering this edge is generated and added to the constraints (4.2) in the main problem.

Another idea is to generate maximal cliques that dominate all of the edge constraints. Balas *et al.* [41] proposes a heuristic that generates maximal cliques in that way. It suggests that this heuristic is very effective in finding good initial solutions. However, we do not use it exactly as it is given in Balas *et al.* [41]. Instead, we use a similar version of it which we call ICG3. ICG3 differs from its original version only in the STEP ADD given in Figure 4.5. An explanation about their difference is also provided in Figure 4.5. We do not report our experiment results on the original one since experiments led us to the conclusion that ICG3 is strongly better than the original one. ICG3 outperformed the original one in both providing better bounds and running time in all of our instances.

INPUT : A graph $G = (V, E)$ with number of vertices n and a clique C

$\mathcal{C} := \emptyset$ (Initially) set of candidate vertices //Find the vertices adjacent to all vertices in clique C

for $i = 1$ to n **do**

  if Vertex $i \notin C$ and $(i, j) \in E \; \forall j \in C$ **then**

    $U := U \cup \{i\}$

  end if

end for

while $U \neq \emptyset$ **do**

  Choose randomly a vertex j from the set U

  $C := C \cup \{j\}$

  for $i \in U$ and $i \neq j$ **do**

    if $(i, j) \notin E$ **then**

      $U := U - \{i\}$

    end if

  end for

  $U := U - \{j\}$

end while

Figure 4.3. MakeMaximal.

```
for i = 1 to n do
  Clique = {i}
  for j = 1 to n do
    if j ≠ i and (j, i) ∈ E then
      U := U ∪ {j}
    end if
  end for
  while U ≠ ∅ do
    Choose randomly a vertex k from the set U
    C := C ∪ {k}
    U := U − {k}
    for j ∈ U do
      if (k, j) ∉ E then
        U := U − {j}
      end if
    end for
  end while
end for
```

Figure 4.4. ICG2.

INPUT :A perfect graph $G = (V, E)$ with set of vertices V and set of edges E

OUTPUT : $\mathcal{C} := \emptyset$ (Initially) set of maximal cliques, $h = 0$ (Initially) length of the set $\mathcal{C}$

for $e \in E$ **do**

// Let e_1 and e_2 denote vertices forming edge e

if $\nexists C_i \in \mathcal{C}$ s.t. $e_1, e_2 \in C_i$ **then**

$h = h + 1$

$C_h := \{e_1, e_2\}$

while $\exists j \in V |$ j is adjacent to all vertices in C_h **do**

STEP ADD: Randomly choose a vertex $v \in V$ from vertices which are adjacent to the vertices in C_h (Original heuristic ([41]) applies the following: Find the vertex $v \in V$ with the highest degree among the vertices which are adjacent to the vertices in C_h)

$C_h := C_h \bigcup \{v\}$

end while

end if

end for

for $i = 1$ to h **do**

Generate a clique constraint from C_i

end for

Figure 4.5. ICG3.

4.2. Preprocessing

Preprocessing strategies can be applied to reduce the number of edges and nodes in the graphs. Removal of edges and nodes reduces the time required to solve the problem. One simple preprocessing strategy is to remove all nodes with degree 0 and all nodes with degree 1 and to add them to the current set of maximum stable set since there has to be at least one maximum stable set containing these nodes. We call this strategy as Simple Preprocessing.

There exists another preprocessing strategy suggested by Rebennack *et al.* [39] which we call Preprocessing2. It is based on the following results. Let us first define $\mathcal{LP}$ as the polytope defined by only all edge inequalities (3.2) without integrality constraints (just with nonnegativity constraints) and consider the following proposition.

Proposition 4.2. *[42] Let x be an extreme point of $\mathcal{LP}$. Then $x_i = 0$, 0.5 or 1 for $1 \leq i \leq n$.*

We do not cover the proof of this proposition. We now present a theorem which uses the result of Proposition 4.2.

Theorem 4.3. *[39] Let y^* be an optimal $(0, \frac{1}{2}, 1)$-valued solution of $\mathcal{LP}$. There is a maximum stable set in graph G that contains $\mathcal{P} := \{v_i \in V | y_i^* = 1\}$.*

Proof. Suppose $\mathcal{P} \neq \emptyset$ (Otherwise proof is trivial). Suppose there is no maximum stable set containing all vertices in $\mathcal{P}$. Consider a maximum stable set S^* of G and define sets of M, N, Z and H as follows:

$$M := \{i \in V | y_i^* = 1\} \bigcap S^*$$

$$N := \{i \in V | y_i^* = 0.5\} \bigcap S^*$$

$$Z := \{i \in V | y_i^* = 0\} \bigcap S^*$$

$$H := \mathcal{P} \backslash M$$

None of the vertices in $\mathcal{P}$ can be adjacent to any vertex in N due to edge inequalities. This implies that $\mathcal{P} \bigcup N$ forms a stable set in G. It follows that $\alpha > |\mathcal{P} \bigcup N|$ by our assumption.

Now, let us construct a new solution y^{new} from y^*. To do that, change only the variables of y^* corresponding to Z and H by making them 1 and keep the others the same. y^{new} is also feasible since none of the vertices in M are adjacent to a vertex in Z. A comparison of the objective function values of y^* and y^{new} gives the following result

$$\sum\nolimits_{i \in V} y_i^* = \alpha < \sum\nolimits_{i \in V} y_i^{new} = \alpha - \frac{|H|}{2} + \frac{|Z|}{2} \tag{4.7}$$

since $\alpha = |M \bigcup Z \bigcup N| > |\mathcal{P} \bigcup N| = |M \bigcup H \bigcup N|$. This contradicts optimality of y^*. Hence, $\mathcal{P}$ is contained in one of the maximum stable sets in G.

□

The preprocessing procedure also uses the result of the following lemma. Rebennack *et al.* [39] presents it for node-weighted graphs. We here present it without node weights.

Lemma 4.4. *[39] Let G be a graph with vertex set V. If there exists a node v_i with the property that its neighborbood, denoted by $N(v_i)$, is a subset of a clique, then there is a maximum stable set in G that contains v_i.*

Proof. Assume there is a vertex v_i with the property described. Denote the set of vertices which are in the maximal clique containing $N(v_i)$ but not in $N(v_i)$ by K.

Define a set of vertices L such that $L := V - (N(v_i) \bigcup K \bigcup \{v_i\})$. Assume no maximal stable set of G contains v_i and consider a maximum stable sets S. If S contains vertices only from set K and L not from set $N(v_i)$, then we can create a new stable set by adding v_i to S and this contradicts with the optimality of S. Note that S cannot contain more than one vertex from set $N(v_i) \bigcup K$ since this set forms a clique. If S contains vertices only from $N(v_i)$ and L, then we can create a new stable set by replacing the vertex of S in $N(v_i)$ with $N(v_i)$. This new stable set is also a maximum stable set of G since it has the same cardinality as S. Hence, there is always a maximum stable in G which contains v_i. □

Rebennack *et al.* [39] suggests using Theorem 4.3 and Lemma 4.4 iteratively as the preprocessing strategy. With Propositon 4.3, nodes with value 1 and 0 in the solution vector are removed as nodes with value 1 are added to the current set of stable set. With Lemma 4.4, once a clique and a vertex with the property described are detected, they are removed from the graph. See [43] for implementation details. We here present only a sketch of the algorithm (See Preprocessing2). Note that we slightly change the preprocessing strategy. The algorithm in Rebennack [43] checks every node with value 1 to remove their neighborhood one by one. However, we use the following observations. First, every vertex with value 0 is adjacent to at least one vertex with value 1. Otherwise, we would obtain a better objective value by increasing the value of that vertex from 0 to 0.5 in the solution vector. Such a solution would be also feasible. Another observation is that no vertex with value 0.5 can be adjacent to a vertex with value 1 since, otherwise, edge inequalities would be violated. It follows that the neighborhood of set of vertices with value 1 is exactly the set of vertices with value 0. Hence, we can just delete the set of vertices with value 0 or 1 from the graph and add the set of vertices with value 1 to the current stable set after solving $\mathcal{LP}$.

4.3. Computational Results

We implemented both Cutting Plane Algorithm and BasicIP formulation in Cplex 12.6.2 built in C++. We applied 3 different strategies (ICG1, ICG2 and ICG3) for the

INPUT : Graph $G = (V, E)$ with set of vertices V and set of edges E

OUTPUT : $S := \emptyset$ (Initially) set of nodes for the maximum stable set, $\bar{G} := \emptyset$ (Initially) the graph after preprocessing

repeat

$Q := \emptyset$, $M := V$, $\bar{G} = G$

while $M \neq \emptyset$ **do**

Step selection: Select a vertex $v \in M$

$M = M - \{v\}$

if $N(v)$ is a clique **then**

$Q = N(v)$

else

goto Step selection

end if

for all $w \in Q$ **do**

$M = M \bigcup (N(w)\ Q)$

end for

$M := M\ Q$

$\bar{G} = \bar{G} \backslash (Q \bigcup \{v\})$

$Q := \emptyset$

$S := S \bigcup \{v\}$

end while

Construct $\mathcal{LP}$ of graph $\bar{G}$ and solve it. Obtain the optimal solution vector x^*

for all $v_i \in V$ **do**

if $x_i^* = 1$ **then**

$S := S \bigcup \{v_i\}$

$\bar{G} = \bar{G} \backslash \{v_i\}$

end if

if $x_i^* = 0$ **then**

$\bar{G} = \bar{G} \backslash \{v_i\}$

end if

end for

until No vertex such that $x_i^* = 1$

return $\bar{G}$ and S

Figure 4.6. Preprocessing2.

initial clique generation of the Cutting Plane Algorithm. We call the Cutting Plane Algorithm with ICG1, the Cutting Plane Algorithm with ICG2 and the Cutting Plane Algorithm with ICG3 by CPA1, CPA2 and CPA3 respectively. All computations were conducted on a Windows 10 PC with a 3.4 GHz Intel Core i7 CPU and 8 GB RAM.

In Chapter 4, we present the Cutting Plane Algorithm which generates 3-cycles and utilize them to detect all violated 3-cycles. However, our experiments revealed that it is very inefficient to employ a strategy checking all 3-cycles for violation since even generating all 3-cycles took about the same time as Cutting Plane Algorithm without 3-cycles in each instance. Consequently, we removed 3-cycles violation search from the Cutting Plane Algorithm. In addition, preprocessing also turned out to be unnecessary since experiments revealed that the Cutting Plane Algorithm performs better without preprocessing. These removals will be justified with the computational results later. Therefore,it is important to note that computational results of CPA1, CPA2 and CPA3 presented here belong to the Cutting Plane Algorithms without preprocessing, 3-cycle generation and control.

To generate perfect graphs instances, we utilized the algorithm PerfectGen of Seker [44]. To the best of our knowledge, PerfectGen is the only algorithm designed to generate solely perfect graphs. It has a built-in diverse set of small-sized perfect graphs to be used in the construction of a perfect graph. It randomly selects some of these small-sized graphs and combine them though perfection-preserving operations to built an end-graph. It takes the number of nodes of the graph to be generated as an input. Hence, we had the freedom of generating as many graphs with the desired size as I wanted.

Table 4.1 and 4.2 reports results of the experiments conducted on 90 instances. We set the time limit of 1200 seconds for each algorithm. Algorithms are terminated without a solution after this time limit is exceeded. In Table 4.1, due to space limitations, each row represents not a graph instance but a set of instances that have the same number of nodes. Each set consists of five graph instances. Experimental results of each instance covered in Table 4.1 can be found individually in Table A.1, A.2 and

A.1. On the other hand, contrary to Table 4.1, each row of Table 4.2 represents a graph instance. They are fully given here since some of the algorithms were terminated before finding a solution. It would not be meaningful to represent them as sets.

Size of the instances varies from very small ones to moderately large ones. Large instances are intended to make inferences about how large instances the Cutting Plane Algorithms are capable of solving within reasonable time limits. On the other hand, small ones enable a comparison between the Cutting Plane and other algorithms, which will be introduced in the following sections. Each column of Table 4.1 and Table 4.2 report a certain characteristic or a computational result of a set of instances and an instance respectively. Each row of Table 4.1 reports the average of the corresponding experimental results for the corresponding set of instances whereas Table 4.2 reports the results for individual instances. The columns of both tables are divided into six groups. The first group Graph represents the characteristics of graphs whereas the other groups inform about experimental results. The group IP reports the results of BasicIP solved by integer programming solver of Cplex. The Groups CPA1, CPA2 and CPA3 reports the results of algorithms CPA1, CPA2 and CPA3 respectively. First column of the table presents names of the instances or set of instances. Next groups $|V|$,$|E|$, Density and $\alpha(G)$ denote the number of vertices, edges, Density and cardinality of a maximum stable set respectively. T columns present running times (CPU times in seconds) of the algorithms integer programming solver of the Cplex, CPA1, CPA2 and CPA3. C columns denote the number of times subproblem (separation) problem is called by the algorithms CPA1, CPA2 and CPA3. EV columns denote number of times main problem is resolved due to edge constraint violation without solving subproblem by the algorithms CPA1 and CPA2. In Table 4.2, we reported the EV and C values of termination moment for the runs terminated without finding a MSS due to time limit 1200 seconds. Hence, it is important to note that these values are not final at that instances.

Comparing the running times in Table 4.1, it can be concluded that all versions of the Cutting Plane Algorithm (CPA1, CPA2 and CPA3) dominantly outperforms IP solver of the Cplex. Additionally, CPA1 performs better than CPA2 and CPA3 on

average. Table 4.2 shows that CPA1 is also better at larger instances. CPA1 solved even some intances with $|E| = 10000$ in less than one minute whereas others failed to terminate in less than 1200 seconds in all of them. Longer CPU times in larger instances can be explained by the separation procedure that consists of solving an IP problem. A close examination of T and C columns of CPA1 in Table 4.2 supports this explanation. Relative inefficiency of CPA3 indicates that it takes relatively long time to generate maximal clique constraints that contains all edges even though algorithm terminated without resorting to the separation problem in the most instances. Success of the Cutting Plane Algorithms can be explained by initial clique generation strategies. We experimented also with other strategies not explained here and their results were not as successful as the ones presented here. When developing the algorithm, we were concerned with the subproblem solved in the Cutting Plane Algorithm. Though it is an NP-hard problem, an examination of C columns reveals that maximum stable set was extracted without resorting to the subproblem in the most instances. Therefore, it is unnecessary to utilize a separation heuristic before solving the separation problem by integer programming. The Cutting Plane Algorithm guarantees returning a maximum stable set when an exact separation is applied.

We also experimented the algorithms with preprocessing and they turned out to be slower. Preprocessing times for each instance is given in Table A.1, A.2, A.3. A examination of T column of Preprocessing in these tables can justify not including preprocessing in the algorithms. Moreover, we present computational results of 3-cycles generation for instances with number of nodes up to 1750 in Table A.4 where columns T and NTC denote generation time in seconds and number of cycles generated respectively. We used $'\mathrm{f}'$ to indicate failure of the generation for an instance. As it can be seen from the table, computation times aggressively rise for instances with number of nodes 1750 and failures can be also seen due to memory. We did not run the generation for larger instances since generation times are considerably larger than the termination times of the Cutting Plane Algorithms even in smaller instances.

Table 4.1. Computational Results 1.

Graph					IP	CPA1			CPA2			CPA3	
Instance Set	$\|V\|$	$\|E\|$	Density	$\alpha(G)$	T	T	C	EV	T	C	EV	T	C
apg1	50	563	0.46	14.2	0.0672	0.014	0	1.6	0.014	0	0.4	0.01	0
apg2	100	1884	0.38	27.8	0.1028	0.014	0	1	0.016	0	0.8	0.02	0.2
apg3	150	6044	0.54	35.8	0.3778	0.018	0	1.8	0.024	0.2	1	0.02	0
apg4	200	9857.8	0.496	46	1.5806	0.018	0	1.6	0.022	0	1	0.02	0
apg5	250	14188	0.454	60.2	4.4118	0.022	0	2	0.036	0	1.8	0.036	0.2
apg6	500	56417.4	0.454	115.2	51.858	0.048	0	2.8	0.09	0.4	1.2	0.092	0
apg7	750	133706	0.478	170.2	31.3294	0.084	0	3	0.1	0	1.4	0.33	0
apg8	1000	251372.2	0.502	208.4	42.2532	0.212	0.2	2.4	0.194	0	1.6	0.736	0.2
apg9	1250	382825.4	0.492	261.4	71.4514	0.394	0.4	2.8	0.788	1	2	1.832	0
apg10	1500	536302	0.478	323.4	87.1396	0.33	0	3	1.264	1	2.2	2.604	0
apg11	1750	787287.6	0.514	362.8	132.9654	0.408	0	2.6	0.834	0	2	4.482	0
apg12	2000	995090.6	0.496	414.4	190.7702	0.538	0	2.2	1.086	0	1.8	8.616	0
apg13	2250	1286000	0.508	469.2	255.9682	1.572	0.6	2.6	2.81	0.8	2	10.716	0.2
apg14	2500	1562000	0.498	517	364.2018	0.952	0	2.4	3.418	0.8	2	11.73	0
apg15	2750	1916000	0.508	537.6	524.657	1.63	0.2	2.8	3.608	0.4	1.8	15.194	0
apg16	3000	2354000	0.524	606.8	718.13	3.126	0.6	2.8	5.12	0.6	2.2	22.784	0
apg17	3250	2616000	0.496	674.2	910.7594	1.802	0	2.8	7.416	1	2	26.496	0
apg18	3500	3146000	0.514	714.4	1183.8548	2.958	0.2	3	9.536	1.2	2.2	37.986	0

Table 4.2. Computational Results 2.

Graph					IP	CPA1			CPA2			CPA3	
Instance	$\|V\|$	$\|E\|$	Density	$\alpha(G)$	T	T	C	EV	T	C	EV	T	C
pg_1_5000	5000	6.29E+10	0.5	1004	>1200	14.4	1	2	26.57	1	3	141.8	0
pg_2_5000	5000	6.76E+11	0.54	1009	>1200	10.87	1	4	45.67	3	3	155.93	0
pg_3_5000	5000	6.24E+11	0.5	1020	>1200	25.17	2	2	24.07	1	2	152.34	0
pg_4_5000	5000	6.40E+11	0.51	1021	>1200	14.73	1	4	14.3	0	2	121	0
pg_5_5000	5000	6.34E+11	0.5	1026	>1200	5.1	0	2	23.6	1	2	137.99	0
pg_1_10000	10000	2.47E+12	0.49	2139	>1200	>1200	1	3	>1200	1	4	>1200	0
pg_2_10000	10000	2.62E+12	0.52	1949	>1200	>1200	1	4	>1200	1	3	>1200	0
pg_3_10000	10000	2.45E+12	0.48	2131	>1200	1120.31	1	5	>1200	1	2	>1200	0
pg_4_10000	10000	2.63E+12	0.53	1957	>1200	38.12	0	3	>1200	1	3	>1200	0
pg_5_10000	10000	2.47E+12	0.49	2013	>1200	48	0	4	>1200	1	3	>1200	0

5. SEMIDEFINITE PROGRAMMING (SDP) FORMULATION

In this section, we present two SDP formulations both of which can compute $\alpha(G)$ in perfect graphs. We first present formulations and results as they apply to a general graph and then we establish the connection with the perfect graphs. Now, before going into the formulations, we introduce some basic facts and notation about SDP.

We denote the set of $n \times n$ real symmetric matrices by S^n. We write $A \succeq 0$ to denote that A is a symmetric and positive semidefinite matrix. A matrix A is a *positive semidefinite* if it satisfies $u^T Au$ for all $u \in \mathbb{R}^n$ or, equivalently, if there exists a matrix $Y \in \mathbb{R}^{n \times n}$ such that $A = Y^T Y$. For a matrix $A \in \mathbb{R}^{n \times n}$, *trace of matrix* A, denoted by $tr(A)$, is defined as follows: $tr(A) = \sum_{k=1}^{n} A_{ii}$. For two matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$, the trace of inner product, denoted by $A \bullet B$ is defined as follows: $A \bullet B = tr(A^T B)$. Standard form of an SDP problem can be given as follows:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & C \bullet X \\ \text{s. t.} \quad & A_j \bullet X = b_j, \ j = 1, 2, ..., m \\ & X \succeq 0. \end{aligned}$$

where $C, A_1, A_2, ..., A_m \in S^n$ and $b_1, b_2, ..., b_m \in \mathbb{R}$ and the dual of this SDP has the following form:

$$\begin{aligned} \underset{y}{\text{maximize}} \quad & b^T y \\ \text{s. t.} \quad & \sum_{i=1}^{m} A_i y_i + S = C \\ & S \succeq 0. \end{aligned}$$

Lovász [45] introduced Lovász's theta number (also known as theta number), denoted by $\vartheta(G)$, to provide bounds for MSS number and chromatic number of a graph. Theta number satisfies the following inequality:

$$\alpha(G) \leq \vartheta(G) \leq \chi(\overline{G})$$

There are several ways to calculate $\vartheta(G)$. Here, we mention two of them which are based on finding the optimal value of an SDP problem.We refer the reader to Grötschel *et al.* [35] for a review of four of these ways with the extension to the weighted case (where a weight is assigned to each node in the objective function). Grötschel *et al.* [35] establishes the equivalence of these of formulations and covers detailed proofs.

The first SDP formulation, which we call SDP1, is due to Lovász [45] and can be given as follows:

$$\begin{aligned} \underset{X}{\text{maximize}} \quad & J \bullet X \\ \text{s. t.} \quad & X_{ij} = 0 \ \ \forall (i,j) \in E \text{ and } I \bullet X = 1 \\ & X \succeq 0. \end{aligned}$$

where J is the matrix of all ones and I is the identity matrix.

After the introduction of SDP1, another SDP formulation to compute $\vartheta(G)$ was proposed by Grötschel *et al.* [35]. Before presenting the formulation, we first define a convex body used in the formulation.

For a graph $G = (V, E)$, a set of unit vectors $\{u_i \in \mathbb{R}^d, i \in V\}$ where d is an arbitrary dimension is said to be an *orthonormal representation* of G if $u_i^T u_j = 0$ for all $(i,j) \notin E$. Let $\{u_i \in \mathbb{R}^d, i \in V\}$ be an orthonormal representation of graph G and let c be the unit vector with the dimension d. Define the following inequality:

$$\sum_{i \in V} (c^T u_i)^2 x_i \leq 1 \tag{5.1}$$

where $x \in \mathbb{R}^{|V|}$ is a vector and x_i represents ith element of vector x. We can now define the theta body of a graph, denoted by $TH(G)$:

$$TH(G) := \{x \in \mathbb{R}^{|V|} \mid x \geq 0 \text{ and } x \text{ satisfies inequality (5.1) for all possible orthonormal representations of } G \text{ and any vector } c\} \quad (5.2)$$

Grötschel *et al.* [46] introduced the theta body and proved the following important theorem:

Theorem 5.1. *[46] Given an arbitrary graph G, the inequality $STAB(G) \subseteq TH(G) \subseteq QSTAB(G)$ holds and the inequality becomes equality if and only if the graph is perfect.*

We can also conclude from Equation (3.10) in Chapter 3 that $STAB(G) = TH(G) = QSTAB(G)$ holds in perfect graphs.

Let us now give the informal proof of Theorem 5.1. Let x be any vector such that $x \in STAB(G)$. Then x can be written as a convex combination of incidence vectors of stable sets in G such that $x = \sum_S \lambda_S \mathcal{X}^S$ and $\sum_S \lambda_S = 1$ where $\mathcal{X}^S$ denotes the incidence vector of stable set S. Consider any stable set S of G. Due to the mutual orthogonality of the vectors $\{u_i \in \mathbb{R}^d, i \in V\}$ and definition of incidence vectors, the following inequality and equality hold:

$$\sum_{i \in S} (c^T u_i)^2 \leq 1 \quad (5.3)$$

$$\sum_{i \in V-S} (c^T u_i)^2 \mathcal{X}_i^S = 0 \quad (5.4)$$

and consequently,

$$\sum_{i \in S} (c^T u_i)^2 \mathcal{X}_i^S \leq 1 \text{ and } \sum_{i \in S} (c^T u_i)^2 \lambda_S \mathcal{X}_i^S \leq \lambda_S$$

Hence, x satisfies Ineqaulity (5.1), which implies that $STAB(G) \subseteq TH(G)$.

For $TH(G) \subseteq QSTAB(G)$, if we show that $TH(G)$ contains all the constraints that define the polytope of $QSTAB(G)$, the proof is complete. Recall that $QSTAB(G)$ contains only the clique constraints. For each clique C, we can choose an arbitrary vector c and construct an orthonormal representation of G such that $c = u_i = u_j$ if $i, j \in C$ and $u_i^T u_j = 0$ if $i \in C$ and $j \notin C$. By plugging this orthonormal representation and c into the inequality 5.1, we obtain

$$\sum_{i \in C} x_i \leq 1$$

Hence, this implies that $TH(G)$ contains all constraints which are necessary to define $QSTAB(G)$; this shows our desired result $TH(G) \subseteq QSTAB(G)$.

Another equivalent definition of $TH(G)$ was proposed by Lovász and Schrijver [47] and can be given as follows:

$$TH(G) := \{x \in \mathbb{R}^{|V|} \mid \exists M = \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \text{ such that } M \succeq 0 \ , \ X_{ii} = x_i \ \ \forall i \in V \\ \text{and } X_{ij} = 0 \ \ \forall (i,j) \in E \text{ where } X \in \mathbb{R}^{|V| \times |V|} \text{ is a matrix}\} \quad (5.5)$$

Any linear objective function over $TH(G)$ can be computed in polynomial-time [35]. Hence, another polynomially solvable SDP formulation, which we will call SDP2, can compute $\vartheta(G)$ or, equivalently, $\alpha(G)$ in perfect graphs:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & e^T x \\ \text{s. t.} \quad & X_{ii} = x_i \ \ \forall i \in V \ , \ X_{ij} = 0 \ \ \forall (i,j) \in E \\ & \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succeq 0. \end{aligned}$$

where e is vector of ones. We know that the following holds for a graph G if and only if G is a perfect graph by Theorem 5.1:

$$TH(G) = \text{conv}\{\mathcal{X}^S \in \mathcal{R}^n \ : \ S \subseteq V \text{ is a stable set}\} \tag{5.6}$$

This also implies $TH(G)$ is a polytope if and only if G is a perfect graph. Though SDP2 can compute the cardinality of a maximum stable set in perfect graphs, it does not ensure finding an maximum stable set. The optimal solution of SDP2 is equivalent to optimal solution of MCliqueIP since both of the problems are optimized over the same polytope.

In addition to SDP1 and SDP2, dual of SDP1 can be used to compute $TH(G)$. Since the SDP solver we used for implementation works on both dual and primal problems simultaneously to be able to solve the primal problem, we also include here the formulation of dual of SDP1 which is used in the algorithm given in the next chapter. The formulation, which we call SDP3, can be given as follows:

$$\begin{aligned} &\min_{\lambda,y,Z} \quad \lambda \\ &\text{s. t.} \quad -\lambda I + Z + \sum_{(i,j)\in E} y_{ij} E_{ij} = -J \qquad Z \succeq 0. \end{aligned}$$

Grötschel *et al.* [6] proposes an approach to extract maximum stable set in perfect graphs by iteratively computing $\alpha(G)$ in smaller subgraphs. It can be summarized as follows: After arbitrarily ordering vertices of the graph, each node is removed one by one. After the removal of each vertex, $\alpha(G)$ is recalculated in the new smaller graph. New graphs are also perfect by the definition of perfect graphs. If $\alpha(G)$ drops by 1, then this implies that the last removed vertex intersects all maximum stable sets in the corresponding new subgraph. Removal of the vertices continues until $\alpha(G)$ drops to 1. When $\alpha(G) = 1$, the vertices whose removal reduced $\alpha(G)$ by one form a maximum stable set together with anyone of the remaining vertices.

Consequently, a maximum stable set of a perfect graph can be found by computing $\alpha(G)$ at most n times. This is the first known polynomial-time algorithm for MSS problem in perfect graphs. This algorithm can extract only one maximum stable set but it may be possible to find other maximum stable sets by reordering the vertices.

In literature, there exists another approach Yıldırım and Fan-Orzechowski [1] which follows a similar but more practical and efficient approach than Grötschel *et al.* [6]. Yıldırım and Fan-Orzechowski [1] proposes an algorithm which utilizes properties of approximate solutions of solved SDP problems presented in the study. The algorithm exploits these properties to efficiently determine a vertex in one of the maximum stable sets. Details of this approach are discussed in the following section. To the best of our knowledge, it is the most efficient polynomial-time algorithm for MSS problem in perfect graphs.

6. AN EFFICIENT ALGORITHM TO EXTRACT A MAXIMUM STABLE SET IN A PERFECT GRAPH THROUGH SDP

Yıldırım and Fan-Orzechowski [1] proposes a more efficient way to extract a maximum stable set similar to Grötschel *et al.* [6]'s algorithm. Since most of the SDP solvers employ primal-dual optimization methods, SDP problems are solved up to a predetermined accuracy in practice. Hence, in practice, setting the accuracy not too small can significantly reduce CPU time of the algorithm. Yıldırım and Fan-Orzechowski [1] establishes that it is enough to solve the SDP1 up to a fairly small accuracy to identify a vertex belonging to a maximum stable set. In this chapter, we present Yıldırım and Fan-Orzechowski [1]'s algorithm along with the results the algorithm designed on. We skip proofs of the results for brevity. We refer the reader to the original paper Yıldırım and Fan-Orzechowski [1] for the proofs. Notation is adopted as given in Yıldırım and Fan-Orzechowski [1] for convenience.

Given a perfect graph $G = (V, E)$, define following sets of stable set

$$\mathcal{S} := \{S \subseteq V : S \text{ is a stable set, } |S| = \alpha(G)\}$$
$$\mathcal{T} := \{T \subseteq V : \text{ is a stable set, } |T| < \alpha(G)\}$$

and following index sets

$$I := \{i \in V : \exists S \in \mathcal{S} \text{ such that } i \in S\}$$
$$J := \{j \in V : j \notin S \text{ for all } S \in \mathcal{S}\}$$

Let ϵ denote the absolute error of solution of an SDP problem which is, indeed, the duality gap between primal and dual solution. A primal-dual approximate solution is said to have an absolute error $\epsilon \geq 0$ if the duality gap is less than ϵ. $\mathcal{X}^S \in \mathcal{R}^n$ is the incidence vector of S where $S \subseteq \{1, ..., n\}$, i.e., $\mathcal{X}_i{}^S = 1$ is $i \in S$ and $\mathcal{X}_i{}^S = 0$ is

$i \notin S$ After defining the necessary notation, we now can establish several properties of near-optimal feasible x which will turn out to be useful in increasing the efficiency of the algorithm.

6.1. Properties of Near-Optimal Feasible Solutions of SDP2

Proposition 6.1. *[1] Let $G = (V, E)$ be a perfect graph and $\epsilon \in [0, 1)$. Suppose that $x \in TH(G)$ satisfies $e^T x \geq \alpha(G) - \epsilon$. Then, $x_j \leq \epsilon$ for all $j \in J$.*

Proof. $x \in TH(G)$ can be written as a convex combination of the incidence vectors of stable sets of G in the following form by using characterization of $TH(G)$ given in Definition (5.6).

$$x = \sum\nolimits_{S \in \mathcal{S}} \lambda_S \mathcal{X}^S + \sum\nolimits_{T \in \mathcal{T}} \lambda_T \mathcal{X}^T \tag{6.1}$$

where $\lambda_S \geq 0$ for all $S \in \mathcal{S}, \lambda_T \geq 0$ for all $T \in \mathcal{T}$, and

$$\sum\nolimits_{S \in \mathcal{S}} \lambda_S + \sum\nolimits_{T \in \mathcal{T}} \lambda_T = 1. \tag{6.2}$$

Then,

$$\sum\nolimits_{T \in \mathcal{T}} \lambda_T \leq \epsilon. \tag{6.3}$$

□

We only include the proof of Proposition 6.1 because we employ a similar proof in the next chapter.

Proposition 6.2. *[1] Any feasible solution X of SDP1 can be transformed into a feasible solution x of SDP2 such that*

$$e^T x \geq J \bullet X$$

where $x_i = \frac{1}{(J \bullet X)X_{ii}}(\sum_{j=1}^{n} X_{ij})^2$ *if* $i \in P$ *and* $x_i = 0$ *otherwise.* $P := \{i \in V : y_i \neq 0\}$, $X = Y^T Y$ *where* Y *can be computed by Cholesky factorization and* y_i *denotes the* i*th column of* Y*.*

Proposition 6.2 indicates that any feasible solution of SDP1 can be transformed into a feasible solution of SDP2 without decrease in the optimal value. Moreover, it is computationally not expensive to apply the transformation.

Corollary 6.3. *[1] Let* $G = (V, E)$ *be a perfect graph and* $\epsilon \in [0, 1)$ *and suppose that* $x \in TH(G)$ *satisfies* $e^T x \geq \alpha(G) - \epsilon$*. Then, set* $\mathcal{K}$ *defined by*

$$\mathcal{K} := \{i \in V : x_i > \epsilon\} \tag{6.4}$$

satisfies $\mathcal{K} \subseteq \mathcal{I}$*.*

Corollary 6.3 implies that we can identify vertices belonging to at least one maximum stale set of a given graph by obtaining an approximate solution of SDP2 up to an error ϵ as long as the solution vector contains an element i such that $x_i \geqslant \epsilon$.

Proposition 6.4. *[1] Let* $G = (V, E)$ *be a graph and let* $S \subseteq V$*,* $S \neq \emptyset$*. Then any feasible solution of the optimization problems SDP1 and SDP2 can be transformed into a feasible solution of each others.*

Proposition 6.5. *Let* $G = (V, E)$ *be a perfect graph. Suppose that* $x \in TH(G)$ *satisfies* $e^T x \geq \alpha(G) - \epsilon$*. If* $\epsilon < \alpha(G)/(n + 1)$*, then the set* $\mathcal{K}$ *defined by expression* (6.4) *is nonempty.*

Proposition 6.5 indicates that $\mathcal{K} \neq \emptyset$ is guaranteed by an SDP2 solution $x \in TH(G)$ such that absolute error of x is less than $\alpha(G)/(n + 1)$. Hence, it is sufficient to solve SDP1 or SDP2 up to such an error to ensure identifying a desired vertex.

In addition to the properties of an approximate solution, Yıldırım and Fan-Orzechowski [1] introduces a way to reduce any feasible solution of SDP1, SDP2 and

SDP3 for a given graph G to a feasible solution of SDP1, SDP2 and SDP3 respectively for a given graph G_S where G_S denotes the subgraph of G induced by the set of vertices $S \subseteq G$. Their algorithm utilizes this reduction as a warm-start strategy by using the reduced solution as a starting point in the next SDP problem to be solved. The following explains this reduction.

Lemma 6.6. *[1] Let $G = (V, E)$ be a perfect graph and X, $\widetilde{x}$ and (λ, y, Z) be any feasible solution of SDP1, SDP2 and SDP3 respectively for graph G. Define a set $S \subseteq G$ and the subgraph G_S of G induced by S. Then, X, $\widetilde{x}$ and (λ, y, Z) can be transformed into a feasible solution of SDP1, SDP2 and SDP3 respectively for G_S.*

We now explain directly how transformations are applied without giving the proof of Lemma 6.6. Define $M := X(S, S)$ to denote the submatrix of X whose rows and columns are indexed by indices in S. Then, $\frac{M}{tr(M)}$ is a feasible solution of SDP1 for G_S if $tr(M) \neq 0$. Otherwise, $\frac{I}{|S|}$ can be used as a reduced solution. Define $\widetilde{x}_S$ to denote the sub-vector of $\widetilde{x}$ restricted to the indices in S. Then, $\widetilde{x}_S \in TH(G_S)$ indicating $\widetilde{x}_S$ is a feasible solution of SDP2 for G_S. Define y_{E_s} to denote the sub-vector of y restricted to the indices $(k, l) \in E_S$. Then, $(\lambda, y_{E_s}, Z(S, S))$ is a feasible solution of SDP3 for G_S.

6.2. The Algorithm of Yıldırım and Fan-Orzechowski [1] (Algorithm1)

We now present the algorithm of Yıldırım and Fan-Orzechowski [1] which we call Algorithm1. It starts with Simple Preprocessing which we also cover in Chapter 4. Remember that the objective of Simple Preprocessing is to remove the vertices with degree 0 or 1 so that the size of the problem may be reduced. Then, the theta problem is solved up to an error less than 1 to determine $\alpha(G)$. After determining $\alpha(G)$, ϵ can be set according to Proposition 6.5. In the first iteration of the while loop, the initial SDP problem is solved again but up to an error less than previous one (0.99). In the iterations other than the first one, the reduced solution obtained from the previous solution (See the last two lines of while loop) is used as a warm start; hence, the new theta problem is solved starting from the reduced solution up to an error ϵ. After solving the theta problem, the resulting approximate solution of SDP1 is transformed

to a solution of SDP3 which is then used to determine a vertex to be added to S. In the determination of a vertex from set K, the one with the maximum number of neighbors is chosen to make the remaining subgraph have less number of vertices, which reduces the size of the problem. Then, the chosen vertex along with its neighbors are removed from the graph G. Lastly, Simple Preprocessing is applied again. The procedure is repeated until no vertices is left in the subgraphs. An advantage of Algorithm1 is the following. SDP1 problem is solved to maximum possible absolute error that ensures finding a vertex contained in at least one of the maximum stable sets. Hence, the time spent on each SDP problem is reduced. Moreover, Algorithm1 also utilizes a warm-start strategy which may considerably reduce the computing time of SDP problem.

6.3. A Variant of Algorithm 1

We now establish that given the solution of SDP3, it is possible to determine more than one vertex contained in the same maximum stable set. Hence, Algorithm1 can be adjusted in a way that it may identify more than one vertex to be added to S instead of adding exactly one vertex to S after obtaining the solution of SDP3. Consequently, it may be possible to lower the number of theta number computations. The following proposition characterize the vertices which are in the same maximum stable stable set.

Proposition 6.7. *Let $G = (V, E)$ be a perfect graph. Suppose that $x \in TH(G)$ satisfies $e^T x \geq \alpha(G) - \epsilon$ and there is a set of vertices M such that $M \subseteq V$ and $|M| = p \geq 2$. Denote the vertices in M by M_i, $i = 1, ..., p$. If the inequality*

$$\sum_{i=1}^{p} x_{M_i} > p - 1 + \epsilon \tag{6.5}$$

holds true, then there exists at least one maximum stable set of G that contains all vertices in M.

Proof. Assume there exist a set $M \subseteq V$ such that it satisfies Inequality 6.5 and there is no maximum stable S in G such that $M \subseteq S$. By equations 6.1 and 6.2, we can

INPUT :A perfect graph $G = (V, E)$ with set of vertices V and set of edges E

OUTPUT : $S := \emptyset$ (Initially) set of nodes for the maximum stable set

Apply the Simple Preprocessing to G and store the vertices to be in the maximum stable set in S.

$//G$ may have been changed since some vertice may have been eleminated.

$\epsilon = 0.99$

Solve SDP1 and SDP3 up to an absolute error ϵ for G. Then, store the solution in (X, y, Z).

$\alpha = C \bullet X$

while $|V| > 0$ **do**

$\epsilon = \alpha/(n+1)$

Starting from (X, y, Z), solve SDP1 and SDP3 up to an absolute error ϵ for G.

Generate a feasible solution of $\widetilde{x}$ of SDP2 for G by Proposition 6.2

Extract the set K defined by Expression (6.4).

$i = \arg\max_{k \in K}\{|N(k)|\}$ where $|N(k)|$ denotes the number of neighbors of vertex k.

$S := S \cup \{i\}$, $V := V - (i \cup N(i))$, $E := E_V$, $\alpha = \alpha - 1$, $G = (V, E)$

Apply the Simple Preprocessing and change $G = (V, E)$ and α accordingly.

Generate feaasible solutions for G from (X, y, Z) according to the Lemma 6.6 to store them in $(X_{new}, y_{new}, Z_{new})$.

$(X, y, Z) = (X_{new}, y_{new}, Z_{new})$

end while

return S

Figure 6.1. Algorithm1.

obtain

$$\sum_{i=1}^{p} x_{M_i} \leq (p-1) \sum\nolimits_{S \in \mathcal{S}} \lambda_S + p \sum\nolimits_{T \in \mathcal{T}} \lambda_T \tag{6.6}$$

since the contribution of any stable set in $\mathcal{S}$ to the sum on the left cannot be more than $(p-1)\lambda_S$ by assumption. Furthermore,

$$(p-1) \sum\nolimits_{Z \in \mathcal{Z}} \lambda_Z + p \sum\nolimits_{T \in \mathcal{T}} \lambda_T \leq (p-1)(1-\epsilon) + p\epsilon = p - 1 + \epsilon \tag{6.7}$$

due to Equality (6.2) and Inequality (6.3).

Inequalities (6.6) and (6.7) simply imply $\sum_{i=1}^{p} x_{M_i} < p - 1 + \epsilon$, contradicting our assumption that M satisfies Inequality (6.5). Hence, there exists a maximum stable set S of G such that $M \subseteq S$ if M satisfies Inequality (6.5). □

To illustrate proposition 6.7 clearly, let us show this proposition for $p = 3$. Suppose $x_{M_1} + x_{M_2} + x_{M_3} > 2 + \epsilon$. By 6.2, x is convex combination of stable set incidence vectors and these vectors can be divided into $2^{p=3}$ categories according to which elements of M they contain. Let λ_{ijk}, $i, j, k \in \{0, 1\}$ denote the sum of coefficients of the incidence vectors in equation 6.1 which contains M_1, M_2, M_3 if $i = 1$, $j = 1$ and $k = 1$ respectively and does not contain otherwise. Since every incidence vector falls into exactly one of these categories, we have $\sum_{i,j,k \in \{0,1\}} \lambda_{ijk} = 1$ by equation 6.2. Additionally, x_1, x_2 and x_3 can be expressed as follows:

$$x_{M_1} = \sum\nolimits_{i=1 \text{ and } j,k \in \{0,1\}} \lambda_{ijk}$$

$$x_{M_2} = \sum\nolimits_{j=1 \text{ and } i,k \in \{0,1\}} \lambda_{ijk}$$

$$x_{M_3} = \sum\nolimits_{k=1 \text{ and } j,k \in \{0,1\}} \lambda_{ijk}$$

When we sum them up, we obtain

$$x_{M_1} + x_{M_2} + x_{M_3} = \lambda_{100} + \lambda_{010} + \lambda_{001} + 2(\lambda_{110} + \lambda_{101} + \lambda_{011}) + 3\lambda_{111}$$

and we know that

$$\lambda_{000} + \lambda_{100} + \lambda_{010} + \lambda_{001} + \lambda_{110} + \lambda_{101} + \lambda_{011} + \lambda_{111} = 1$$

Hence, if we assume none of the maximum stable sets in G contains these 3 vertices, then $\lambda_{111} \leq \sum_{T \in \mathcal{T}} \lambda_T \leq \epsilon$. Therefore, we obtain

$$2 + \epsilon = 2 + \lambda_{111} = 2 - 2\lambda_{111} + 3\lambda_{111} =$$
$$2(\lambda_{000} + \lambda_{100} + \lambda_{010} + \lambda_{001} + \lambda_{110} + \lambda_{101} + \lambda_{011}) + 3\lambda_{111} \geq x_{M_1} + x_{M_2} + x_{M_3},$$

which contradicts with our assumption. Hence, there exists a maximum stable set containing these three vertices in M.

We can propose a different version of Algorithm1, which we call Algorithm2, based on the Proposition 6.7. Algorithm2 differs from Algorithm1 in the way it analyzes the solution of SDP2 for the identification of a vertex to be added to S. Contrary to Algorithm1 which adds only one vertex to S from an SDP3 solution, Algorithms2 can add more than one vertex by utilizing Proposition 6.7. Moreover, ϵ is a constant in Algorithm2 as opposed to Algorithm1 which determines it dynamically. Different approaches can be adopted in the determination of ϵ. As the value of ϵ decreases at least up to a limit, it is possible that Algorithm2 can extract more vertices that can be added to the maximum stable set. On the other hand, computational time of an SDP problem is non-decreasing with an increasing ϵ value. There is a tradeoff between obtaining the solution of SDP2 up to a larger error and finding more vertices satisfying Inequality 6.7. Therefore, the value of ϵ may have an important affect on the performance of the Algorithm2. We use $\epsilon = 0.1$ in our implementation. Algorithm2 is presented in Figure 6.2.

In Algorithm1, only the vertex i is added to S whereas, in Algorithm2, a set of vertices R is added. When $p = 1$, we can employ Algorithm2 in the same way as Algorithm1 is employed by choosing the vertex with the highest number of neighbors from the set K.

6.4. New SDP Formulations to Solve MSS Problem

Algorithm1 and Algorithm2 extract a maximum stable set in perfect graphs by solving more than one SDP problem. They both solve the theta problems in the smaller subgraphs and progressively construct a maximum stable. Contrary to these algorithms which solve multiple SDP problem, in this section, we present an SDP formulation which can directly solve the MSS problem in perfect graphs. However, it comes at very high cost of having to solve it up to a very small error and SDP4 turns out to be very inefficient and impossible to solve in practise even at small instances.

The formulation, which we call SDP4, can be given as follows:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & z^T x \\ \text{s. t.} \quad & X_{ii} = x_i \ \ \forall i \in V \ , \ X_{ij} = 0 \ \ \forall (i,j) \in E \\ & \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succeq 0. \end{aligned}$$

where z is a vector of dimension $|V|$ such that

$$z_i = 1 + 2^{n-i}\delta \text{ and } \sum_{i=1}^{N-1} 2^i \delta \ = 1 \tag{6.8}$$

The only difference between SDP2 and SDP4 is their objectives. The unbalanced coefficients of z ensures obtaining an optimal extreme point of $TH(G)$ when solved up to a very small error instead of convex combinations of optimal extreme points. Though the error polynomially grows with increasing V, SDP4 can be still solved

INPUT :A perfect graph $G = (V, E)$ with set of vertices V and set of edges E

OUTPUT : $S := \emptyset$ (Initially) set of nodes for the maximum stable set

Apply the Simple Preprocessing to G and store the vertices to be in the maximum stable set in S.

$//G$ may have been changed since some vertice may have been eleminated.

while $|V| > 0$ **do**

  if this is the first iteration **then**

    Solve SDP1 and SDP3 without a starting point up to an absolute error $\epsilon = 0.1$ for G. Then, store the solution in (X, y, Z). Calculate $\alpha = C \bullet X$

  else

    Starting from (X, y, Z), solve SDP1 and SDP3 up to an absolute error $\epsilon = 0.1$ for G. Then, store the solution in (X, y, Z).

  end if

  Generate a feasible solution of $\widetilde{x}$ of SDP2 for G by Proposition 6.2

  Order the vertices of G in decreasing order according to their values in $\widetilde{x}$ and store them in the set L in this order.

  $p = 1$

  while $\sum_{i=1}^{p} \widetilde{x}_{L_i} > p - 1 + \epsilon$ **do**

    $p = p + 1$

  end while

  if $p \neq 1$ **then**

    Define a new set R containing first $p - 1$ vertices in L

    $S := S \cup R$, $V := V - (R \cup N(R))$, $E := E_V$, $\alpha = \alpha - p$, $G = (V, E)$

  else

    Employ the vertex selection process for S and graph reduction as they are applied in Algorithm1

  end if

  Apply the Simple Preprocessing and change the $G = (V, E)$ and α accordingly.

  Transform (X, y, Z) to feaasible solutions of new G according to Lemma 6.6

end while

return S

Figure 6.2. Algorithm2.

in polynomial-time. The approximation algorithm of Grötschel *et al.* [35] based on Ellipsoid method computes the solution of an SDP problem up to any accuracy ϵ in time that is polynomial in $\log \frac{1}{\epsilon}$.

Proposition 6.8. *Let $G = (V, E)$ be a perfect graph. Denote by x^* the optimal solution of SDP4 for G. Suppose that $x \in TH(G)$ such that $z^T x \geq z^T x^* - \epsilon$ where $\epsilon < \frac{\delta}{2}$. Then, $S := \{i \in V | x_i^* > 0.5\}$ is a maximum stable set of G.*

Proof. Let us construct a maximum stable S^* of G in the following way: First, label vertices with the same indices as their indices in the vector z. Remember the definitions of sets $\mathcal{S}$, $\mathcal{T}$, I and J from Chapter 6. Choose the vertex $v \in I$ with the smallest index number and add v to S^*. Then, remove the vertices that have smaller index number than vertex v or that are adjacent to vertex v from the graph G. For the remaining graph, apply the same procedure keeping the order of indices. At the end of this procedure, we obtain a maximum stable set of G. S^* is constructed together with Equation 6.8 to ensure the following characterization of S^*:

$$z^T \mathcal{X}_{S^\star} - z^T \mathcal{X}_S > \delta \quad \forall S \in \mathcal{S} - \{S^\star\} \tag{6.9}$$

where $\mathcal{X}_{S_\star}$ and $\mathcal{X}_S$ are incidence vectors of corresponding stable sets. This inequality holds since $2^i > \sum_{j=1}^{i-1} 2^j \delta$ implies choosing the vertex with the lowest possible index always yields highest contribution to the objective function no matter how rest of the maximum stable set follows. Hence, it ensures optimality by choosing always the vertex with the lowest possible index.

By Equation 6.8, the following also holds:

$$z^T \mathcal{X}_{S^\star} - z^T \mathcal{X}_T > \delta \quad \forall T \in \mathcal{T} \tag{6.10}$$

where $\mathcal{X}_T$ is the incidence vector of the corresponding stable set since $z^T \mathcal{X}_T < \alpha(G)$ and $z^T \mathcal{X}_{S^\star} > \alpha(G) + \delta$ for $n \geq 2$.

Therefore, $\mathcal{X}_{S^\star}$ is the optimal solution of SDP4 for G. By our assumption of x, we obtain

$$z^T x^* \geq z^T \mathcal{X}_{S^\star} - \epsilon > z^T \mathcal{X}_{S^\star} - \frac{\delta}{2} \tag{6.11}$$

and x can be expressed in the following form since $TH(G)$ is a polytope:

$$x = \lambda_{S^*} \mathcal{X}^{S^*} + \sum\nolimits_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z \mathcal{X}^Z \tag{6.12}$$

where $\lambda_{S^*} + \sum_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z = 1$. By Inequalities (6.9) and (6.10), we know that

$$\sum\nolimits_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z z^T \mathcal{X}^Z \leq \sum\nolimits_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z (z^T \mathcal{X}^{S^*} - \delta) \tag{6.13}$$

When this inequality is combined with Inequality (6.11), we obtain

$$z^T \mathcal{X}_{S^\star} - \frac{\delta}{2} < z^T x^* \leq z^T \mathcal{X}_{S^\star} - \delta \sum\nolimits_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z \tag{6.14}$$

which implies $\frac{\delta}{2} > \delta \sum_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z$ and consequently, we obtain

$$\frac{1}{2} > \sum\nolimits_{Z \in \mathcal{S} \cup \mathcal{T} - S^*} \lambda_Z \tag{6.15}$$

which implies that any vertex $v \notin S^*$ cannot satisfy that $x_v > 0.5$ whereas any vertex $v \in S^*$ satisfies $x_v > 0.5$ □

Proposition 6.8 indicates that a maximum stable set in perfect graphs can be extracted in polynomial time by solving a single SDP formulation. Although SDP4 is not efficient in practice due to the error decreasing polynomially with increasing number of vertices, it theoretically proves that exactly one SDP formulation can solve MSS problem in perfect graphs. Moreover, considering a computer can handle numbers up to a certain accuracy, it may be impossible to solve SDP4 for graphs larger than a certain size.

Therefore, another method to apply this approach of assigning different weights to the vertices in the objective function can be the following way. We determine an objective function weight for each vertex in a way similar to expression 6.8 but using $z_i = 1+(n-i)\delta$ instead and then, we can incorporate the result of Proposition 6.7 into the algorithm. By using these coefficients instead of the ones in Expression 6.8, we can have a lower ϵ but this does not guarantee finding a maximum stable set in exactly one iteration. It increases the probability that we can find it in exactly one iteration, resulting in a fewer number of SDP problems solved. However, it still uses considerably smaller ϵ compared to Algorithm1, which makes this method computationally much less efficient than Algorithm1. The corresponding formulation, which we call SDP5, can be given as follows:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & z^T x \\ \text{s. t.} \quad & X_{ii} = x_i \ \ \forall i \in V \ , \ X_{ij} = 0 \ \ \forall (i,j) \in E \\ & \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succeq 0. \end{aligned}$$

where z is a vector of dimension $|V|$ such that

$$z_i = 1 + (n-i)\delta \text{ and } \sum_{i=1}^{N-1} i\delta \ = 1 \tag{6.16}$$

We now give two propositions which we use in Algorithm3.

Proposition 6.9. *Let $G = (V, E)$ be a perfect graph and $E \neq \emptyset$ .Denote by x^* the optimal solution of SDP5 for G. Suppose that $x \in TH(G)$ such that $z^T x \geq z^T x^* - \epsilon$ where $\epsilon = \delta$. Then, $\alpha = \lfloor z^T x \rfloor$*

Proof. By our assumption, we have

$$z^T x^* \leq z^T x + \epsilon \tag{6.17}$$

and G is not empty , which implies

$$z^T x^* \geq \alpha(G) + \delta \tag{6.18}$$

By combining these two inequalities (6.17) and (6.18), we obtain $z^T x \geq \alpha(G)$ Moreover, $z^T x < \alpha(G) + 1$ trivially holds. □

Proposition 6.5 does not hold for SDP5 since the objective is different. We propose the following proposition to adjust Proposition 6.5 for SDP5.

Proposition 6.10. *Let $G = (V, E)$ be a perfect graph. Denote by x^* the optimal solution of SDP5 for G. Suppose that $x \in TH(G)$ such that $z^T x \geq z^T x^* - \epsilon$ where $\epsilon = \delta$ and there is a set of vertices M such that $M \subseteq V$ and $|M| = p \geq 2$. Denote the vertices in M by M_i, $i = 1, ..., p$. If the inequality*

$$\sum_{i=1}^{p} x_{M_i} > p - 1 + 2\epsilon \tag{6.19}$$

holds true, then there exists at least one maximum stable set of G that contains all vertices in M.

Proof. Recall Equations (6.1) and (6.2):

$$x = \sum\nolimits_{S \in \mathcal{S}} \lambda_S \mathcal{X}^S + \sum\nolimits_{T \in \mathcal{T}} \lambda_T \mathcal{X}^T$$

$$\sum\nolimits_{S \in \mathcal{S}} \lambda_S + \sum\nolimits_{T \in \mathcal{T}} \lambda_T = 1$$

This expression of x is still valid for SDP5 since feasible regions is the same. Plugging the expression of x into $z^T x \geq z^T x^* - \epsilon$, we obtain

$$z^T x^* - \epsilon \leq \sum_{S \in \mathcal{S}} \lambda_S z^T \mathcal{X}^S + \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T \leq \sum_{S \in \mathcal{S}} \lambda_S z^T x^* + \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T \tag{6.20}$$

$$z^T x^* - \epsilon \leq \sum_{S \in \mathcal{S}} \lambda_S z^T \mathcal{X}^S + \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T \leq \sum_{S \in \mathcal{S}} \lambda_S z^T x^* + \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T \tag{6.21}$$

When terms are rearranged,

$$z^T x^* - \sum_{S \in \mathcal{S}} \lambda_S z^T x^* - \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T = (1 - \sum_{S \in \mathcal{S}} \lambda_S) z^T x^* - \sum_{T \in \mathcal{T}} \lambda_T z^T \mathcal{X}^T \leq \epsilon \tag{6.22}$$

and finally we obtain

$$\sum_{T \in \mathcal{T}} \lambda_T (z^T \mathcal{X}^* - z^T \mathcal{X}^T) \leq \epsilon \tag{6.23}$$

Now, let us examine $(z^T x^* - z^T \mathcal{X}^T)$ to find a lower bound for it. Assume $\alpha \neq 1$. Clearly,

$$z^T \mathcal{X}^* - z^T \mathcal{X}^T \geq min(z^T \mathcal{X}^*) - max(z^T \mathcal{X}^T) \tag{6.24}$$

The max and min values can be found by assigning the maximum possible weights and minimum possible weights respectively. Hence, we obtain

$$min(z^T x^*) = \alpha(G) + \frac{\alpha(G)(\alpha(G) - 1)\delta}{2} \tag{6.25}$$

and since a stable set which is not maximum can have at most $\alpha(G) - 1$ vertices,

$$min(z^T x^*) = \alpha(G) + \frac{\alpha(G)(\alpha(G) - 1)\delta}{2} \tag{6.26}$$

Therefore,

$$min(z^T x^*) - max(z^T \mathcal{X}^T) = \frac{n^2 - n}{2} + \frac{2n + 2\alpha(G)^2 - 2\alpha(G) - 2\alpha(G)n}{2} = \quad (6.27)$$

$$1 - (\alpha(G) - 1)(n - \alpha(G))\delta = 1 - 2\frac{(\alpha(G) - 1)(n - \alpha(G))}{(n-1)n} \quad (6.28)$$

To find the minimum of this, we take the derivative of $(\alpha(G) - 1)(n - \alpha(G))$ with respect to $\alpha(G)$:

$$n - 2\alpha_* + 1 = 0 \quad (6.29)$$

which implies $\alpha_* = \frac{n+1}{2}$. Plugging α_* into the equation, we obtain

$$min(z^T \mathcal{X}^*) - max(z^T \mathcal{X}^T) = \delta\frac{n(n-1)}{2} - \delta\frac{(n-1)^2}{4} = \frac{(n-1)(n+1)}{2(n-1)n} > 0.5 \quad (6.30)$$

Inequalities 6.23 and 6.30 together imply

$$2\epsilon > \sum\nolimits_{T \in \mathcal{T}} \lambda_T \quad (6.31)$$

Therefore, in Inequality (6.5), we put 2ϵ instead of ϵ and finally, we obtain

$$(p-1)\sum\nolimits_{Z \in \mathcal{Z}} \lambda_Z + n\sum\nolimits_{T \in \mathcal{T}} \lambda_T \leq (p-1)(1 - 2\epsilon) + p2\epsilon = p - 1 + 2\epsilon \quad (6.32)$$

□

We can now present the algorithms which utilizes the results given throughout this section. The algorithm, which we call Algorithm3 is given in Figure 6.4.

INPUT :A perfect graph $G = (V, E)$ with set of vertices V and set of edges E

OUTPUT : $S := \emptyset$ (Initially) set of nodes for the maximum stable set

while $|V| > 0$ **do**

Apply the Simple Preprocessing to G and add the resulting vertices to S.

Solve SDP5 and and its dual up to an absolute error $\epsilon = \delta$ for G and start the solution from $(x_{new}, y_{new}, Z_{new})$ if it is not null. Then, store the solution in (x, y, Z) where (y, Z) constitutes the dual solution.

Set $\alpha = \lfloor z^T x \rfloor$ by Proposition 6.9

Order the vertices of G in decreasing order according to their values in x and store them in set L in this order. $p = 1$

while $p \leq |V|$ **do**

if $\sum_{i=1}^{p} x_{L_i} > p - 1 + 2\epsilon$ **then**

$p = p + 1$

else

break

end if

end while

if $p \neq 1$ **then**

Define a new set R containing first $p - 1$ vertices in L

$S := S \cup R$, $V := V - (R \cup N(R))$, $E := E_V$, $\alpha = \alpha - p$, $G = (V, E)$

else

Employ the vertex selection process for S and graph reduction as they are implemented in Algorithm1

end if

Apply the Simple Preprocessing and change the $G = (V, E)$ and α accordingly.

Generate feaasible solutions for new G from (x, y, Z) according to Lemma 6.6 to store them in $(x_{new}, y_{new}, Z_{new})$.

// Lemma 6.6 does not cover the reduction of dual of SDP2 but it can be done similar to SDP3.

end while

return S

Figure 6.3. Algorithm3.

6.5. Implementation

We used the same computer as in Chapter 4 for our computational experiments. We implemented Algorithm1, Algorithm2 and Algorithm3 in R 3.3.2 where an R package called Rcsdp is used to solve SDP problems in the algorithms. Rcsdp is an R interface between R and CSDP version 6.1.1. CSDP (Borchers [48]) is an SDP library that implements a predictor corrector variant of the semidefinite programming algorithm of Helmberg *et al.* [49]. The function csdp of Rcsdp is called to compute the theta function. This function solves a given SDP problem up to prespecified accuracy that can be adjusted by the control argument. However, it allows for only relative error and all algorithms need SDP problems to be solved up to an absolute error. Additionally, it does not enable us to predetermine a starting point for the problem, which is necessary for the warm-start strategy in the algorithms. Therefore, we modified Rcsdp in a way it enables us to use a predetermined starting point and a prespecified absolute error as a termination criteria. These slight modifications were only intended to provide more options for the user. We did not change the structure of the algorithm.

Instances used in the experiments are also the same as Chapter 4. We run the algorithms without imposing a time limit for the instances with number of nodes up to 250. For the instances with number of nodes higher than 250, the algorithm failed to find a MSS due to an error resulting from SDP solver. Hence, we here present computational results of only the instances with number of nodes less than or equal to 250. Computational results are presented in Table 6.1. Each row of the table belongs to a graph instance. The columns are divided into four groups. Columns of the first group Graph reports characteristics of the corresponding graph instance except the column T which presents CPU time of the first theta problem solved (up to $\epsilon = 0.99$) in Algorithm1 in seconds. The column T, in fact, can be interpreted as the minimum time needed to find $\alpha(G)$. Hence, we include it in the group Graph. The second, third and fourth groups reports computational results for Algorithm1, Algorithm2 and Algorithm3 respectively where columns T denote the CPU time in seconds and columns Iterations denote number of times while loop is executed (i.e., number of times theta problem is solved).

Table 6.1. SDP Computational Results.

Graph						Algorithm1		Algorithm2		Algorithm3	
Instance	$\|V\|$	$\|E\|$	*density*	α	T	T	*Iterations*	T	*Iterations*	T	*Iterations*
pg_1_50	50	634	0.52	16	0.48	0.79	2	0.66	2	1	1
pg_2_50	50	935	0.76	9	1.15	1.72	2	1.44	2	2.42	1
pg_3_50	50	382	0.31	16	0.15	0.39	3	0.25	2	0.31	1
pg_4_50	50	228	0.19	17	0.05	0.31	6	0.19	2	0.14	1
pg_5_50	50	636	0.52	13	0.4	0.7	2	0.91	9	0.94	1
pg_1_100	100	1641	0.33	33	5.54	6.59	6	6.68	1	13.5	1
pg_2_100	100	1693	0.34	28	6.66	9.71	6	8.59	5	18.94	1
pg_3_100	100	3048	0.62	16	32.67	44.77	3	41.14	3	75.83	1
pg_4_100	100	1486	0.3	32	5.07	6.87	7	6.47	3	12.15	1
pg_5_100	100	1552	0.31	30	4.36	6.69	6	5.87	4	11.44	1
pg_1_150	150	8137	0.73	24	730.64	879.06	4	874.44	4	1608.3	1
pg_2_150	150	6645	0.59	30	396.89	480.21	5	477.72	3	927.2	1
pg_3_150	150	4164	0.37	43	96.61	129.12	7	121.22	3	254.89	1
pg_4_150	150	4943	0.44	42	167.16	169.89	9	182.71	3	379.8	1
pg_5_150	150	6331	0.57	40	379.49	382.32	7	414.92	5	810.22	1
pg_1_200	200	13109	0.66	40	3466.56	4336.89	5	3752.7	4	7815.61	1
pg_2_200	200	5016	0.25	60	190.93	247.65	18	226.33	5	522.27	1
pg_3_200	200	14132	0.71	33	4532.2	5670.98	5	4916.71	8	9028.44	1
pg_4_200	200	9745	0.49	45	1371.21	1624.57	7	1621.6	11	3290.28	1
pg_5_200	200	7287	0.37	52	578.53	740.1	15	684.45	4	1469.97	1
pg_1_250	250	11169	0.36	70	2246.39	2816.85	16	2605.19	3	5176.08	1
pg_2_250	250	13968	0.45	64	4461.42	5216.52	9	5082.32	8	10367.3	1
pg_3_250	250	13506	0.43	69	4066.37	4751.17	17	4593.06	2	9384.43	1
pg_4_250	250	13500	0.43	53	4044.72	4729.14	14	4254.54	2	9695.68	1
pg_5_250	250	18797	0.6	45	10122.11	12895.77	8	11539.88	9	25045.2	1

Computational results reveal that it took considerably long time to solve the first theta problem in the instances with number of nodes more than 100. Considering the instances with number of nodes 200 and 250, it can be concluded that density together with number of nodes is an important factor that considerably affects the running time. When running times are considered, Algorithm2 performed better than the others on average though the difference between it and Algorithm1 is small. Difference is higher in the larger instances. When Algorithm1 and Algorithm2 is compared, Algorithm1 performed considerably better than Algorithm3 in all instances though Algorithm3 was able to find a maximum stable set in one iteration in all instances. This difference can be explained by the structure of Algorithm1 which enables utilization of a smaller size subgraph at the first time it is possible to reduce the graph. On the contrary, Algorithm2 constantly runs on the same large graph.

The solution of theta problem by the csdp function in Rcsdp package of R consists of the largest part of the running time. Rcsdp includes just a small amount of code written in R and calls the functions from CSDP that is written in C. Therefore, algorithms are run mostly through C in the background though we implemented them in R.

7. COMPARISON OF THE ALGORITHMS

In this chapter, we compare a cutting plane and an SDP-based algorithm we covered in the previous sections with the branch-and-bound algorithm, MaxCliqueDyn, based on subgraph coloring and proposed by Konc and Janezic [50]. We used the new version of MaxCliqueDyn (See its website for details.) in our experiments. To the best of our knowledge, there exists no branch-and-bound algorithm specifically designed for perfect graphs. Hence, we picked an algorithm that can solve MSS problem in general graphs. Wu and Hao [7] presents computational comparison of many powerful brach-and-bound algorithms for MSS problem and its comparison reveals that MaxCliqueDyn is one of the most efficient algorithms. Therefore, it sets good example of a powerful branch-and-bound algorithm for our computational comparison.

Comparison results are given in Table 7.1, 7.2, 7.3 and 7.4. For comparison, we picked CPA1 as Cutting Plane Algorithm and Algorithm2 as SDP-based algorithm since they performed better than others on average. Algorithm2 is capable of solving only very small instances within reasonable time limits compared to the others even though it is the only polynomial algorithm. Algorithm2 performed considerably worse than the others. Moreover, it was not able to solve the instances with number of nodes 500 while CPA1 solved some instances with number of nodes 10000 in less than one minute. This implies a polynomial-time solvable algorithm does not necessarily indicate being efficient in practice.

MaxCliqueDyn performed close to CPA1 at instances with number of nodes up to 1000. However, it performed poorly at larger instances though there are a few instances it performed well. MaxCliqueDyn did not turn out to be more efficient than the IP solver of Cplex at larger instances. IP solver of Cplex solved only instances with number of nodes up to 3500 in less than 20 minutes. On the other hand, CPA1 was able to solve instances with number of nodes up to 5000 in less than 30 seconds. It solved even some larger instances in less than 1 minute.

Table 7.1. Comparison of Algorithms 1.

Graph					IP	CPA1	Algorithm2	MaxCliqueDyn
Instance	$\|V\|$	$\|E\|$	*density*	α	T	T	T	T
pg_1_50	50	634	0.52	16	0.09	0.02	0.66	0.01
pg_2_50	50	935	0.76	9	0.06	0.01	1.44	0.01
pg_3_50	50	382	0.31	16	0.06	0.01	0.25	0.01
pg_4_50	50	228	0.19	17	0.06	0.01	0.19	0.01
pg_5_50	50	636	0.52	13	0.07	0.02	0.91	0.01
pg_1_100	100	1641	0.33	33	0.07	0.01	6.68	0.01
pg_2_100	100	1693	0.34	28	0.13	0.02	8.59	0.01
pg_3_100	100	3048	0.62	16	0.19	0.01	41.14	0.01
pg_4_100	100	1486	0.30	32	0.06	0.01	6.47	0.01
pg_5_100	100	1552	0.31	30	0.07	0.02	5.87	0.01
pg_1_150	150	8137	0.73	24	0.45	0.01	874.44	0.02
pg_2_150	150	6645	0.59	30	0.55	0.02	477.72	0.01
pg_3_150	150	4164	0.37	43	0.27	0.02	121.22	0.01
pg_4_150	150	4943	0.44	42	0.33	0.02	182.71	0.02
pg_5_150	150	6331	0.57	40	0.30	0.02	414.92	0.01
pg_1_200	200	13109	0.66	40	2.79	0.01	3752.7	0.02
pg_2_200	200	5016	0.25	60	0.33	0.01	226.33	0.01
pg_3_200	200	14132	0.71	33	2.41	0.02	4916.71	0.03
pg_4_200	200	9745	0.49	45	1.44	0.02	1621.6	0.03
pg_5_200	200	7287	0.37	52	0.94	0.03	684.45	0.02
pg_1_250	250	11169	0.36	70	1.87	0.02	2605.19	0.04
pg_2_250	250	13968	0.45	64	2.19	0.02	5082.32	0.02
pg_3_250	250	13506	0.43	69	4.91	0.02	4593.06	0.02
pg_4_250	250	13500	0.43	53	2.88	0.02	4254.54	0.02
pg_5_250	250	18797	0.60	45	10.21	0.03	11539.88	0.03
pg_1_500	500	48266	0.39	132	29.16	0.04	f	0.06
pg_2_500	500	40930	0.33	133	66.81	0.05	f	3.93
pg_3_500	500	64652	0.52	102	38.93	0.06	f	0.04
pg_4_500	500	76248	0.61	87	50.38	0.05	f	0.04
pg_5_500	500	51991	0.42	122	74.01	0.04	f	0.05

Table 7.2. Comparison of Algorithms 2.

Graph					IP	CPA1	Algorithm2	MaxCliqueDyn
Instance	$\|V\|$	$\|E\|$	*density*	α	T	T	T	T
pg_4_500	500	76248	0.61	87	50.38	0.05	f	0.04
pg_5_500	500	51991	0.42	122	74.01	0.04	f	0.05
pg_1_750	750	139901	0.50	163	32.85	0.08	f	0.12
pg_2_750	750	115531	0.41	205	27.21	0.10	f	0.18
pg_3_750	750	150484	0.54	154	28.78	0.07	f	0.15
pg_4_750	750	131752	0.47	170	35.11	0.08	f	0.14
pg_5_750	750	130862	0.47	159	32.70	0.09	f	0.17
pg_1_1000	1000	292081	0.58	180	44.19	0.15	f	0.19
pg_2_1000	1000	236317	0.47	227	40.80	0.15	f	4.57
pg_3_1000	1000	230818	0.46	236	45.99	0.14	f	0.36
pg_4_1000	1000	219774	0.44	198	34.93	0.51	f	4.34
pg_5_1000	1000	277871	0.56	201	45.35	0.11	f	94.60
pg_1_1250	1250	418758	0.54	231	64.95	0.62	f	>1200
pg_2_1250	1250	341511	0.44	265	85.96	0.26	f	>1200
pg_3_1250	1250	372454	0.48	273	64.36	0.67	f	636.15
pg_4_1250	1250	374945	0.48	262	69.07	0.21	f	>1200
pg_5_1250	1250	406459	0.52	276	72.92	0.21	f	>1200
pg_1_1500	1500	537356	0.48	332	85.22	0.28	f	1.48
pg_2_1500	1500	581510	0.52	297	110.25	0.35	f	>1200
pg_3_1500	1500	535781	0.48	303	88.00	0.34	f	>1200
pg_4_1500	1500	462418	0.41	368	72.65	0.36	f	230.16
pg_5_1500	1500	564445	0.50	317	79.58	0.32	f	>1200
pg_1_1750	1750	845955	0.55	330	117.34	0.42	f	>1200
pg_2_1750	1750	743604	0.49	358	140.29	0.41	f	>1200
pg_3_1750	1750	806527	0.53	339	164.11	0.40	f	164.38
pg_4_1750	1750	707500	0.46	411	121.63	0.39	f	>1200
pg_5_1750	1750	832852	0.54	376	121.47	0.42	f	295.17
pg_1_2000	2000	898351	0.45	439	188.28	0.52	f	>1200
pg_2_2000	2000	1.08E+06	0.54	367	176.26	0.59	f	3.46
pg_3_2000	2000	1.09E+06	0.54	398	187.67	0.51	f	>1200
pg_4_2000	2000	887102	0.44	480	179.90	0.54	f	>1200
pg_5_2000	2000	1.02E+06	0.51	388	221.74	0.53	f	>1200

Table 7.3. Comparison of Algorithms 3.

Graph					IP	CPA1	Algorithm2	MaxCliqueDyn
Instance	$\|V\|$	$\|E\|$	*density*	α	T	T	T	T
pg_1_2250	2250	1.24E+06	0.49	497	240.84	0.71	f	>1200
pg_2_2250	2250	1.45E+06	0.57	415	341.98	1.99	f	>1200
pg_3_2250	2250	1.29E+06	0.51	482	199.84	0.70	f	>1200
pg_4_2250	2250	1.23E+06	0.49	462	227.03	3.74	f	>1200
pg_5_2250	2250	1.22E+06	0.48	490	270.16	0.72	f	>1200
pg_1_2500	2500	1.64E+06	0.52	516	311.95	1.07	f	>1200
pg_2_2500	2500	1.54E+06	0.49	509	450.42	0.92	f	>1200
pg_3_2500	2500	1.63E+06	0.52	513	370.40	0.91	f	>1200
pg_4_2500	2500	1.49E+06	0.48	538	407.22	0.92	f	>1200
pg_5_2500	2500	1.51E+06	0.48	509	281.02	0.94	f	>1200
pg_1_2750	2750	1.96E+06	0.52	511	440.34	3.53	f	>1200
pg_2_2750	2750	1.95E+06	0.52	560	532.09	1.18	f	>1200
pg_3_2750	2750	2.00E+06	0.53	497	495.64	1.06	f	>1200
pg_4_2750	2750	1.93E+06	0.51	513	658.22	1.25	f	>1200
pg_5_2750	2750	1.74E+06	0.46	607	497.01	1.13	f	>1200
pg_1_3000	3000	2.44E+06	0.54	553	780.00	1.43	f	>1200
pg_2_3000	3000	2.38E+06	0.53	644	732.48	6.92	f	>1200
pg_3_3000	3000	2.29E+06	0.51	621	650.08	4.36	f	>1200
pg_4_3000	3000	2.32E+06	0.52	610	736.80	1.40	f	>1200
pg_5_3000	3000	2.34E+06	0.52	606	691.30	1.52	f	>1200
pg_1_3250	3250	2.60E+06	0.49	719	1276.61	1.77	f	>1200
pg_2_3250	3250	2.69E+06	0.51	632	760.51	1.71	f	>1200
pg_3_3250	3250	2.36E+06	0.45	706	635.83	1.74	f	>1200
pg_4_3250	3250	2.89E+06	0.55	592	870.84	1.94	f	>1200
pg_5_3250	3250	2.54E+06	0.48	722	1010.01	1.85	f	>1200
pg_1_3500	3500	3.55E+06	0.58	653	1629.26	2.03	f	>1200
pg_2_3500	3500	2.92E+06	0.48	752	1053.63	2.33	f	>1200
pg_3_3500	3500	2.95E+06	0.48	760	1117.12	6.12	f	>1200
pg_4_3500	3500	3.12E+06	0.51	740	942.10	2.30	f	>1200
pg_5_3500	3500	3.19E+06	0.52	667	1177.16	2.01	f	>1200

Table 7.4. Comparison of Algorithms 4.

Graph					IP	CPA1	Algorithm2	MaxCliqueDyn
Instance	$\|V\|$	$\|E\|$	*density*	α	T	T	T	T
pg_1_5000	5000	6.29E+10	0.50	1004	>1200	14.40	f	>1200
pg_2_5000	5000	6.76E+11	0.54	1009	>1200	10.87	f	>1200
pg_3_5000	5000	6.24E+11	0.50	1020	>1200	25.17	f	>1200
pg_4_5000	5000	6.40E+11	0.51	1021	>1200	14.73	f	>1200
pg_5_5000	5000	6.34E+11	0.50	1026	>1200	5.10	f	>1200
pg_1_10000	10000	2.47E+12	0.49	2139	>1200	>1200	f	>1200
pg_2_10000	10000	2.62E+12	0.52	1949	>1200	>12000	f	>1200
pg_3_10000	10000	2.45E+12	0.48	2131	>1200	1120.31	f	>1200
pg_4_10000	10000	2.63E+12	0.53	1957	>1200	38.12	f	>1200
pg_5_10000	10000	2.47E+12	0.49	2013	>1200	48.00	f	>1200

8. CONCLUSION

In this thesis, we studied the maximum stable set (MSS) problem in perfect graphs. MSS problem in general graphs is an NP-hard problem and it has been studied due to both theoretical and practical interest, which invoked development of various approaches. On the other hand, when MSS problem is considered only in perfect graphs, only a limited number of studies can be found and it is polynomially solvable by an SDP-based algorithm. However, it is known that SDP solvers are slow in practice. To the best of our knowledge, there has been no study computationally comparing an SDP-based algorithm with other exact approaches for MSS problem in perfect graphs. Considering these facts, objective of our thesis is to compare the performances of algorithms that can solve MSS problem in perfect graphs. Therefore, we designed some algorithms or implemented some of the existing algorithms and tested the efficiency of these algorithms on perfect graph instances.

Our thesis began with a trivial IP formulation of the MSS problem. Then, we presentes an LP formulation with possibly an exponential number of constraints which can solve MSS problem in perfect graphs based on structural properties of polytopes of perfect graphs. Following this LP formulation, to solve the MSS problem in perfect graphs, we introduced Cutting Plane Algorithms which differs from each other only in the initial clique generation.

In Chapter 6, we presented an SDP-based algorithm, Algorithm1, which can solve MSS problem in perfect graphs. We introduced a proposition leading to a different version of Algorithm1, Algorithm2, which turned out to provide a minor improvement to the SDP-based algorithm. We then presented an SDP formulation SDP4 which solves MSS problem but with an error growing exponentially with the size of the graph. Lastly, following a similar approach we used in SDP4, we presented another version of Algorithm1, Algorithm3, which turned out to be considerably slower than Algorithm1.

In Chapter 7, we compared CPA1 (the most efficient Cutting Plane Algorithm on average), Algorithm2 (the most efficient SDP-based algorithm on average) with a branch-and-bound algorithm called MaxCliqueDyn. The comparison revealed that CPA1 is capable of solving much larger insatnces than the others while Algorithm2 is capable of solving only very small instances in a reasonable time limit.

As a future research, one can focus on developing separation procedures which don't depend on an IP formulation since an efficient separation procedure can lead the cutting plane algorithms to be able to solve MSS problem on much larger graphs. Additionally, a branch-and-bound algorithm may be designed to exploit properties of perfect graphs, leading to a more efficient branch-and-bound algorithm for the solution of MSS problem in perfect graphs. Apart from MSS problem, coloring problem can also be solved in polynomial time in perfect graphs through a SDP-based algorithm [6]. A future research can compare performances of different algorithms for coloring problem in perfect graphs as we do for MSS problem.

REFERENCES

1. Yıldırım, A. and X. Fan-Orzechowski, "On extracting maximum stable sets in perfect graphs using Lovász's theta function", *Computational Optimization and Applications*, Vol. 33, No. 3, pp. 229–247, 3 2006.

2. Berge, C., *Graphs and hypergraphs*, New York: Elsevier, 1973.

3. Lovász, L., "Normal hypergraphs and the perfect graph conjecture", *Discrete Mathematics*, pp. 253–267, 1979.

4. Chudnovsky, M., N. Robertson, P. Seymour and R. Thomas, "The strong perfect graph theorem", *Annals of Mathematics*, Vol. 164, p. 51–229, 1979.

5. Garey, M. and D. S. Johnson, *Computers and intractability*, Vol. 29, W.H. Freeman New York, 2002.

6. Grötschel, M., L. Lovász and A. J. Schrijver, "Polynomial algorithms for perfect graphs", *Annals of Discrete Mathematics*, pp. 325–326, 1984.

7. Wu, Q. and J.-K. Hao, "A review on algorithms for maximum clique problems", *European Journal of Operational Research*, Vol. 242, No. 3, pp. 693–709, 2015.

8. Boginski, V., S. Butenko and P. M. Pardalos, "Mining market data: A network approach", *Computers & Operations Research*, Vol. 33, No. 11, pp. 3171–3184, 2006.

9. Brotcorne, L., G. Laporte and F. Semet, "Fast heuristics for large scale covering-location problems", *Computers & Operations Research*, Vol. 29, No. 6, pp. 651–665, 2002.

10. Balasundaram, B. and S. Butenko, "Graph domination, coloring and cliques in

telecommunications", *Handbook of Optimization in Telecommunications*, pp. 865–890, Springer, 2006.

11. Wu, Q. and J.-K. Hao, "A review on algorithms for maximum clique problems", *European Journal of Operational Research*, Vol. 242, No. 3, pp. 693–709, 2015.

12. Tomita, E. and T. Kameda, "An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments", *Journal of Global optimization*, Vol. 37, No. 1, pp. 95–111, 2007.

13. Li, C. M. and Z. Quan, "An efficient branch-and-bound algorithm based on maxSAT for the maximum clique problem", *AAAI*, Vol. 10, pp. 128–133, 2010.

14. Tomita, E., Y. Sutani, T. Higashi, S. Takahashi and M. Wakatsuki, "A simple and faster branch-and-bound algorithm for finding a maximum clique", *International Workshop on Algorithms and Computation*, pp. 191–203, Springer, 2010.

15. San Segundo, P., D. Rodríguez-Losada and A. Jiménez, "An exact bit-parallel algorithm for the maximum clique problem", *Computers & Operations Research*, Vol. 38, No. 2, pp. 571–581, 2011.

16. Babel, L. and G. Tinhofer, "A branch and bound algorithm for the maximum clique problem", *ZOR Methods and Models of Operations Research*, Vol. 34, p. 207–217, 1990.

17. Balas, E. and C. S. Yu, "Finding a Maximum clique in an arbitrary graph", *Siam J. Comput.*, Vol. 15, p. 1054–1068, 1986.

18. Östergård, P. R. J., "A fast algorithm for the maximum clique problem", *Discrete Applied Mathematics*, Vol. 120, p. 197–207, 2002.

19. Babel, L. and G. Tinhofer, "A branch and bound algorithm for the maximum clique problem", *ZOR Methods and Models of Operations Research*, Vol. 34, pp.

207–217, 1990.

20. Rebennack, S., G. Reinelt and P. M. Pardalos, "A tutorial on branch and cut algorithms for the maximum stable set problem", *International Transactions in Operational Research*, Vol. 19, No. 1-2, pp. 161–199, 2012.

21. Warrier, D., W. Wilhelm, J. Warren and I. Hicks, "A branch-and-price approach for the maximum weight independent set problem", *Networks*, Vol. 46, No. 4, p. 198–209, 2005.

22. Carargan, R. and P. M. Pardalos, "An exact algorithm for the maximum clique problem", *Operations Research Letters 9*, pp. 375–382, 1990.

23. Pardalos, P. M. and A. T. Phillips, "A global optimization approach for solving the maximum clique problem", *International Journal of Computer Mathematics*, Vol. 33, p. 209–216, 1990.

24. Bourjolly, J.-M., G. Laporte and H. Mercure, "A combinatorial column generation algorithm for the maximum stable set problem", *Operations research letters*, Vol. 20, No. 1, pp. 21–29, 1997.

25. Bomze, I. M., M. Budinich, P. M. Pardalos and M. Pelillo, "The maximum clique problem", *Handbook of combinatorial optimization*, pp. 1–74, Springer, 1999.

26. Giandomenico, M., A. N. Letchford, F. Rossi and S. Smriglio, "A new approach to the stable set problem based on ellipsoids", *International Conference on Integer Programming and Combinatorial Optimization*, pp. 223–234, Springer, 2011.

27. Golumbic, M. C., *Algorithmic graph theory and perfect graphs*, Vol. 57, Elsevier, 2004.

28. Eisenbrand, F., S. Funke, N. Garg and J. Könemann, "A combinatorial algorithm for computing a maximum independent set in at-perfect graph", *Proceedings of the*

fourteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 517–522, Society for Industrial and Applied Mathematics, 2003.

29. Minty, G. J., "On maximal independent sets of vertices in claw-free graphs", *Journal of Combinatorial Theory, Series B*, Vol. 28, pp. 284–304, 1980.

30. Mosca, R., "Polynomial algorithms for the maximum stable set problem on particular classes of P5-free graphs", *Information Processing Letters*, Vol. 61, pp. 137–143, 1997.

31. Simone, C. D. and A. Sassano, "Stability number of bull- and chair- free graphs", *Discrete Applied Mathematics*, Vol. 41, pp. 121–129, 1993.

32. Arora, S. and S. Kale, "A combinatorial, primal-dual approach to semidefinite programs", *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pp. 227–236, ACM, 2007.

33. Bouwmans, T., F. Porikli, B. Höferlin and A. Vacavant, *Background Modeling and Foreground Detection for Video Surveillance*, CRC press, 2014.

34. Padberg, M. W., "On the facial structure of set packing polyhedra", *Mathematical programming*, Vol. 5, No. 1, pp. 199–215, 1973.

35. Grötschel, M., L. Lovász and A. J. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Vol. 2 of *Algorithms and Combinatorics*, Springer, 1988.

36. Padberg, M. and G. Rinaldi, "Optimization of a 532 city symmetric traveling salesman problem by branch and cut", *Operations Research Letters*, Vol. 6, pp. 1 – 7, 1987.

37. Khachiyan, L. G., "Polynomial algorithms in linear programming", *USSR Computational Mathematics and Mathematical Physics*, Vol. 20, No. 1, pp. 53–72, 1980.

38. Grötschel, M., L. Lovász and A. Schrijver, "The ellipsoid method and its conse-

quences in combinatorial optimization.", *Combinatorica*, pp. 169 – 197, 1981.

39. Rebennack, S., M. Oswald, D. O. Theis, H. Seitz, G.Reinelt and P. Pardalos, "A branch and cut solver for the maximum stable set problem", *Journal of Combinatorial Optimization*, Vol. 21, No. 4, pp. 434–457, 2011.

40. Nemhauser, G. L. and L. A. Wolsey, "Integer and combinatorial optimization. Interscience series in discrete mathematics and optimization", *ed: John Wiley & Sons*, 1988.

41. Balas, E., S. Ceria, G. Cornuéjols and G. Pataki, "Polyhedral methods for the maximum clique problem", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 26, pp. 11–28, 1996.

42. Nemhauser, G. L. and L. E. Trotter, "Properties of vertex packing and independence system polyhedra", *Mathematical Programming*, Vol. 6, No. 1, pp. 48–61, 1974.

43. Rebennack, S., "Maximum stable set problem: A branch cut solver", *Diplomarbeit, Ruprecht–Karls Universität Heidelberg, Heidelberg, Germany*, 2006.

44. Seker, O., *The selective graph coloring problem*, Tech. rep., Department of Industrial Engineering, Bogazici University, 2017.

45. Lovász, L., "On the Shannon capacity of a graph", *IEEE Transactions on Information Theory*, Vol. 25, pp. 1 – 7, 1979.

46. Grötschel, M., L. Lovász and A. Schrijver, "Relaxations of vertex packing", *Journal of Combinatorial Theory, Series B*, Vol. 40, No. 3, pp. 330–343, 1986.

47. Lovász, L. and A. Schrijver, "Cones of matrices and set-functions and 0–1 optimization", *SIAM Journal on Optimization*, Vol. 1, No. 2, pp. 166–190, 1991.

48. Borchers, B., "CSDP, AC library for semidefinite programming", *Optimization*

methods and Software, Vol. 11, No. 1-4, pp. 613–623, 1999.

49. Helmberg, C., F. Rendl, R. J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming", *SIAM Journal on Optimization*, Vol. 6, No. 2, pp. 342–361, 1996.

50. Konc, J. and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem", *Proteins*, Vol. 4, No. 5, 2007.

APPENDIX A: COMPUTATIONAL RESULTS

These tables contains the all instances summarized in Table 4.1. In addition to the groups in Table 4.1, they includes a group called Preprocessing with two columns. Preprocessing reports results of the Preprocessing Algorithm Preprocessing2 given in Section 4.2. RV denotes number of removed vertices by preprocessing algorithm and T denotes running time of the algorithm in seconds.

Table A.1. Detailed Computational Results 1.

Graph					IP	Preprocessing		CPA1			CPA2			CPA3	
Instance	$\|V\|$	$\|E\|$	Density	$\alpha(G)$	T	T	RV	T	C	EV	T	C	EV	T	C
pg _1_50	50	634	0.52	16	0.09	0.01	12	0.02	0	2	0.01	0	0	0.01	0
pg _2_50	50	935	0.76	9	0.06	0.01	6	0.01	0	2	0.01	0	0	0.01	0
pg _3_50	50	382	0.31	16	0.06	0.01	19	0.01	0	1	0.02	0	1	0.01	0
pg _4_50	50	228	0.19	17	0.06	0.01	20	0.01	0	1	0.01	0	0	0.01	0
pg _5_50	50	636	0.52	13	0.07	0.02	10	0.02	0	2	0.02	0	1	0.01	0
pg _1_100	100	1641	0.33	33	0.07	0.01	21	0.01	0	1	0.02	0	1	0.04	1
pg _2_100	100	1693	0.34	28	0.13	0.01	41	0.02	0	2	0.02	0	2	0.01	0
pg _3_100	100	3048	0.62	16	0.19	0.05	8	0.01	0	1	0.01	0	0	0.02	0
pg _4_100	100	1486	0.30	32	0.06	0.01	7	0.01	0	0	0.01	0	0	0.01	0
pg _5_100	100	1552	0.31	30	0.07	0.01	15	0.02	0	1	0.02	0	1	0.02	0
pg _1_150	150	8137	0.73	24	0.45	0.04	0	0.01	0	1	0.02	0	1	0.02	0
pg _2_150	150	6645	0.59	30	0.55	0.05	17	0.02	0	3	0.02	0	1	0.02	0
pg _3_150	150	4164	0.37	43	0.27	0.04	27	0.02	0	2	0.02	0	2	0.02	0
pg _4_150	150	4943	0.44	42	0.33	0.07	14	0.02	0	2	0.05	1	1	0.02	0
pg _5_150	150	6331	0.57	40	0.30	0.03	2	0.02	0	1	0.01	0	0	0.02	0
pg _1_200	200	13109	0.66	40	2.79	0.05	1	0.01	0	1	0.03	0	2	0.02	0
pg _2_200	200	5016	0.25	60	0.33	0.03	2	0.01	0	1	0.02	0	1	0.02	0
pg _3_200	200	14132	0.71	33	2.41	0.08	4	0.02	0	2	0.02	0	1	0.02	0
pg _4_200	200	9745	0.49	45	1.44	0.04	1	0.02	0	2	0.02	0	0	0.02	0
pg _5_200	200	7287	0.37	52	0.94	0.04	11	0.03	0	2	0.02	0	1	0.02	0
pg _1_250	250	11169	0.36	70	1.87	0.05	0	0.02	0	2	0.02	0	1	0.02	0
pg _2_250	250	13968	0.45	64	2.19	0.05	27	0.02	0	1	0.03	0	1	0.03	0
pg _3_250	250	13506	0.43	69	4.91	0.05	0	0.02	0	1	0.02	0	1	0.03	0
pg _4_250	250	13500	0.43	53	2.88	0.05	36	0.02	0	2	0.03	0	2	0.07	1
pg _5_250	250	18797	0.60	45	10.21	0.07	0	0.03	0	4	0.08	0	4	0.03	0
pg _1_500	500	48266	0.39	132	29.16	0.19	0	0.04	0	2	0.04	0	1	0.09	0
pg _2_500	500	40930	0.33	133	66.81	0.16	4	0.05	0	4	0.06	0	2	0.07	0
pg _3_500	500	64652	0.52	102	38.93	0.24	0	0.06	0	4	0.15	1	1	0.11	0
pg _4_500	500	76248	0.61	87	50.38	0.33	10	0.05	0	2	0.06	0	1	0.11	0
pg _5_500	500	51991	0.42	122	74.01	0.20	40	0.04	0	2	0.14	1	1	0.08	0
pg _1_750	750	139901	0.50	163	32.85	0.65	21	0.08	0	3	0.12	0	2	0.28	0
pg _2_750	750	115531	0.41	205	27.21	0.46	1	0.10	0	4	0.08	0	1	0.50	0
pg _3_750	750	150484	0.54	154	28.78	0.68	13	0.07	0	2	0.12	0	2	0.32	0
pg _4_750	750	131752	0.47	170	35.11	0.60	0	0.08	0	3	0.09	0	1	0.31	0
pg _5_750	750	130862	0.47	159	32.70	0.59	19	0.09	0	3	0.09	0	1	0.24	0

Table A.2. Detailed Computational Results 2.

Graph					IP	Preprocessing		CPA1			CPA2			CPA3	
Instance	$\|V\|$	$\|E\|$	Density	$\alpha(G)$	T_1	T_2	RV	T_3	C_1	EV_1	T_4	C_2	EV_2	T_5	C_3
pg _1_1000	1000	292081	0.58	180	44.19	1.43	9	0.15	0	3	0.20	0	1	0.69	0
pg _2_1000	1000	236317	0.47	227	40.80	1.01	0	0.15	0	2	0.19	0	2	0.59	0
pg _3_1000	1000	230818	0.46	236	45.99	0.98	37	0.14	0	2	0.15	0	1	0.59	0
pg _4_1000	1000	219774	0.44	198	34.93	2.16	48	0.51	1	3	0.20	0	2	1.06	1
pg _5_1000	1000	277871	0.56	201	45.35	1.22	11	0.11	0	2	0.23	0	2	0.75	0
pg _1_1250	1250	418758	0.54	231	64.95	1.76	0	0.62	1	3	1.26	2	2	1.08	0
pg _2_1250	1250	341511	0.44	265	85.96	1.53	0	0.26	0	4	0.33	0	3	1.69	0
pg _3_1250	1250	372454	0.48	273	64.36	1.55	24	0.67	1	2	1.30	2	2	2.60	0
pg _4_1250	1250	374945	0.48	262	69.07	3.19	26	0.21	0	3	0.31	0	2	2.07	0
pg _5_1250	1250	406459	0.52	276	72.92	1.71	2	0.21	0	2	0.74	1	1	1.72	0
pg _1_1500	1500	537356	0.48	332	85.22	4.69	13	0.28	0	2	0.46	0	1	2.48	0
pg _2_1500	1500	581510	0.52	297	110.25	2.68	0	0.35	0	4	1.24	1	3	2.64	0
pg _3_1500	1500	535781	0.48	303	88.00	2.47	32	0.34	0	2	0.53	0	2	3.50	0
pg _4_1500	1500	462418	0.41	368	72.65	2.09	2	0.36	0	4	1.26	1	2	1.93	0
pg _5_1500	1500	564445	0.50	317	79.58	2.45	1	0.32	0	3	2.83	3	3	2.47	0
pg _1_1750	1750	845955	0.55	330	117.34	3.75	11	0.42	0	3	0.97	0	2	5.75	0
pg _2_1750	1750	743604	0.49	358	140.29	3.28	0	0.41	0	2	0.76	0	2	3.73	0
pg _3_1750	1750	806527	0.53	339	164.11	3.54	18	0.40	0	3	0.88	0	2	5.18	0
pg _4_1750	1750	707500	0.46	411	121.63	3.11	5	0.39	0	2	0.69	0	2	3.75	0
pg _5_1750	1750	832852	0.54	376	121.47	3.71	10	0.42	0	3	0.87	0	2	4.00	0
pg _1_2000	2000	898351	0.45	439	188.28	4.00	10	0.52	0	2	0.81	0	1	5.23	0
pg _2_2000	2000	1.08E+06	0.54	367	176.26	4.92	0	0.59	0	3	1.29	0	2	13.25	0
pg _3_2000	2000	1.09E+06	0.54	398	187.67	4.94	10	0.51	0	2	1.21	0	2	8.65	0
pg _4_2000	2000	887102	0.44	480	179.90	4.16	0	0.54	0	2	0.96	0	2	5.82	0
pg _5_2000	2000	1.02E+06	0.51	388	221.74	4.77	37	0.53	0	2	1.16	0	2	10.13	0
pg _1_2250	2250	1.24E+06	0.49	497	240.84	5.70	5	0.71	0	2	3.22	1	2	7.94	0
pg _2_2250	2250	1.45E+06	0.57	415	341.98	6.70	13	1.99	1	3	1.97	0	2	11.10	0
pg _3_2250	2250	1.29E+06	0.51	482	199.84	5.89	10	0.70	0	3	3.13	1	2	9.95	0
pg _4_2250	2250	1.23E+06	0.49	462	227.03	5.97	64	3.74	2	3	2.45	1	1	16.63	1
pg _5_2250	2250	1.22E+06	0.48	490	270.16	5.55	4	0.72	0	2	3.28	1	3	7.96	0
pg _1_2500	2500	1.64E+06	0.52	516	311.95	7.68	7	1.07	0	3	2.03	0	2	10.14	0
pg _2_2500	2500	1.54E+06	0.49	509	450.42	7.16	13	0.92	0	2	1.87	0	2	12.01	0
pg _3_2500	2500	1.63E+06	0.52	513	370.40	8.12	1	0.91	0	2	7.97	3	2	13.60	0
pg _4_2500	2500	1.49E+06	0.48	538	407.22	7.39	0	0.92	0	2	1.83	0	2	11.47	0
pg _5_2500	2500	1.51E+06	0.48	509	281.02	14.37	24	0.94	0	3	3.39	1	2	11.43	0
pg _1_2750	2750	1.96E+06	0.52	511	440.34	9.45	0	3.53	1	3	2.98	0	2	15.79	0
pg _2_2750	2750	1.95E+06	0.52	560	532.09	9.13	0	1.18	0	3	5.05	1	2	17.24	0
pg _3_2750	2750	2.00E+06	0.53	497	495.64	19.61	26	1.06	0	2	2.60	0	1	14.23	0
pg _4_2750	2750	1.93E+06	0.51	513	658.22	10.01	1	1.25	0	4	2.65	0	2	14.98	0
pg _5_2750	2750	1.74E+06	0.46	607	497.01	8.87	1	1.13	0	2	4.76	1	2	13.73	0

Table A.3. Detailed Computational Results 3.

Graph					IP	Preprocessing		CPA1			CPA2			CPA3	
Instance	$\|V\|$	$\|E\|$	Density	$\alpha(G)$	T	T	RV	T	C	EV	T	C	EV	T	C
pg_1_3000	3000	2.44E+06	0.54	553	780.00	12.87	0	1.43	0	3	6.36	1	2	24.05	0
pg_2_3000	3000	2.38E+06	0.53	644	732.48	11.75	8	6.92	2	2	3.24	0	2	23.40	0
pg_3_3000	3000	2.29E+06	0.51	621	650.08	11.99	1	4.36	1	3	3.16	0	2	23.13	0
pg_4_3000	3000	2.32E+06	0.52	610	736.80	12.03	17	1.40	0	3	6.64	1	3	18.85	0
pg_5_3000	3000	2.34E+06	0.52	606	691.30	26.51	16	1.52	0	3	6.20	1	2	24.49	0
pg_1_3250	3250	2.60E+06	0.49	719	1276.61	13.84	16	1.77	0	2	7.30	1	2	27.05	0
pg_2_3250	3250	2.69E+06	0.51	632	760.51	14.24	4	1.71	0	3	6.48	1	2	28.63	0
pg_3_3250	3250	2.36E+06	0.45	706	635.83	12.45	0	1.74	0	2	7.31	1	2	23.36	0
pg_4_3250	3250	2.89E+06	0.55	592	870.84	14.38	2	1.94	0	4	4.70	0	2	30.15	0
pg_5_3250	3250	2.54E+06	0.48	722	1010.01	12.44	0	1.85	0	3	11.29	2	2	23.29	0
pg_1_3500	3500	3.55E+06	0.58	653	1629.26	20.74	46	2.03	0	2	13.31	2	2	46.19	0
pg_2_3500	3500	2.92E+06	0.48	752	1053.63	15.53	16	2.33	0	3	13.18	2	3	26.04	0
pg_3_3500	3500	2.95E+06	0.48	760	1117.12	15.66	1	6.12	1	3	4.52	0	2	36.87	0
pg_4_3500	3500	3.12E+06	0.51	740	942.10	38.32	28	2.30	0	4	8.83	1	2	43.72	0
pg_5_3500	3500	3.19E+06	0.52	667	1177.16	17.32	16	2.01	0	3	7.84	1	2	37.11	0

Table A.4. 3-cycle Generation Results.

Instance	$\|V\|$	$\|E\|$	Density	T	NTC	Instance	$\|V\|$	$\|E\|$	Density	T	NTC
pg_1_50	50	634	0.52	0.001	4138	pg_1_750	750	139901	0.5	6.6	1.96E+07
pg_2_50	50	935	0.76	0.003	9451	pg_2_750	750	115531	0.41	5.113	1.55E+07
pg_3_50	50	382	0.31	0.001	1942	pg_3_750	750	150484	0.54	5.067	1.55E+07
pg_4_50	50	228	0.19	0.001	474	pg_4_750	750	131752	0.47	17.731	5.44E+07
pg_5_50	50	636	0.52	0.002	4911	pg_5_750	750	130862	0.47	12.236	3.73E+07
pg_1_100	100	1641	0.33	0.006	16641	pg_1_1000	1000	292081	0.58	11.103	3.35E+07
pg_2_100	100	1693	0.34	0.007	20548	pg_2_1000	1000	236317	0.47	12.334	3.76E+07
pg_3_100	100	3048	0.62	0.02	61975	pg_3_1000	1000	230818	0.46	16.07	4.87E+07
pg_4_100	100	1486	0.3	0.004	10374	pg_4_1000	1000	219774	0.44	33.082	9.31E+07
pg_5_100	100	1552	0.31	0.004	12458	pg_5_1000	1000	277871	0.56	22.288	6.38E+07
pg_1_150	150	8137	0.73	0.094	274474	pg_1_1250	1250	418758	0.54	25.676	7.55E+07
pg_2_150	150	6645	0.59	0.058	185093	pg_2_1250	1250	341511	0.44	26.046	7.61E+07
pg_3_150	150	4164	0.37	0.028	84533	pg_3_1250	1250	372454	0.48	27.921	8.41E+07
pg_4_150	150	4943	0.44	0.034	103306	pg_4_1250	1250	374945	0.48	44.447	1.28E+08
pg_5_150	150	6331	0.57	0.048	145552	pg_5_1250	1250	406459	0.52	65.636	1.53E+08
pg_1_200	200	13109	0.66	0.177	509054	pg_1_1500	1500	537356	0.48	45.248	1.30E+08
pg_2_200	200	5016	0.25	0.024	73451	pg_2_1500	1500	581510	0.52	65.636	1.53E+08
pg_3_200	200	14132	0.71	0.202	617045	pg_3_1500	1500	535781	0.48	59.03	1.38E+08
pg_4_200	200	9745	0.49	0.108	300611	pg_4_1500	1500	462418	0.41	34.253	93920557
pg_5_200	200	7287	0.37	0.059	172766	pg_5_1500	1500	564445	0.5	59.03	137784210
pg_1_250	250	11169	0.36	0.146	452770	pg_1_1750	1750	845955	0.55	f	f
pg_2_250	250	13968	0.45	0.18	544010	pg_2_1750	1750	743604	0.49	913.197	1.91E+08
pg_3_250	250	13506	0.43	0.295	901324	pg_3_1750	1750	806527	0.53	3605.88	2.60E+08
pg_4_250	250	13500	0.43	1.018	2.97E+06	pg_4_1750	1750	707500	0.46	1843.84	2.70E+08
pg_5_250	250	18797	0.6	0.733	2.15E+06	pg_5_1750	1750	832852	0.54	f	f
pg_1_500	500	48266	0.39	1.665	5.10E+06						
pg_2_500	500	40930	0.33	2.348	7.27E+06						
pg_3_500	500	64652	0.52	1.129	3.44E+06						
pg_4_500	500	76248	0.61	5.76	1.77E+07						
pg_5_500	500	51991	0.42	3.536	1.06E+07						