

RICE UNIVERSITY

Clique Generalizations and Related Problems

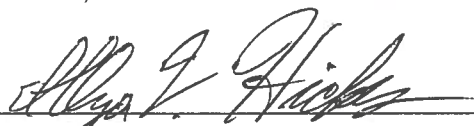
by

Cynthia Ivette Wood

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:



Illya V. Hicks, Chair
Professor of Computational and Applied
Mathematics



Swarat Chaudhuri
Associate Professor of Computer Science



Andrew J. Schaefer
Noah Harding Chair
Professor of Computational and Applied
Mathematics



Yin Zhang
Professor of Computational and Applied
Mathematics

Houston, Texas

November, 2015

ABSTRACT

Clique Generalizations and Related Problems

by

Cynthia Ivette Wood

A large number of real-world problems can be model as optimization problems in graphs. The clique model was introduced to aid the study of network structure for social interaction. Each vertex represented an actor and the edges represented the relations between them. Nevertheless, the model has been shown to be restrictive for modeling real-world problems, since it leaves out subgraphs that do not have all possible edges. As a consequence, clique generalizations were introduced to overcome the disadvantages of the clique model. In this thesis, I present three computationally difficult combinatorial optimization problems related to clique generalization problems: co-2-plexes and k -cores.

A k -core is a subgraph with minimum degree greater than or equal to k . In this work, I discuss the minimal k -core problem and the minimum k -core problem. I present a backtracking algorithm to find all minimal k -cores of a given undirected graph and its applications to the study of associative memory. The proposed method is a modification of the Bron and Kerbosch algorithm for finding all cliques of an undirected graph. In addition, I study the polyhedral structure of the k -core polytope. The minimum k -core problem is modeled as a binary integer program and relaxed as a linear program. Since the relaxation yields to a non-integral solution, cuts must be added in order to improve the solution. I show that edge and cycle transversals of

the graph give valid inequalities for the convex hull of k -cores.

A set of pairwise non-adjacent vertices defines a stable set. A stable set is the complement of a clique. A co-2-plex is a subgraph with degree less than or equal to one, and it is a stable set relaxation. I introduce a study of the maximum weighted co-2-plex (MWC2P) problem for $\{\text{claw}, \text{bull}\}$ -free graphs and present two polynomial time algorithms to solve it. One of the algorithms transforms the original graph to solve an instance of the maximum weighted stable set problem utilizing Minty's algorithm. The second algorithm is an extension of Minty's algorithm and solves the problem in the original graph.

All the algorithms discussed in this thesis were implemented and tested. Numerical results are provided for each one of them.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Illya Hicks, who has been a truly outstanding mentor over the past years. Dr. Hicks' advice and support have been key elements in my success as a graduate student. I especially admire his ability to encourage and empower me and the rest of our research group to achieve our maximum potential.

Secondly, I would like to thank the rest of my committee: Dr. Andrew Schaefer, Dr. Yin Zhang and Dr. Swarat Chaudhuri for their guidance and support. Especial thanks go to Dr. Richard Tapia for mentoring me all these years. I also want to acknowledge Dr. Steve Cox and Dr. Kathryn Hedrick for their mentorship and ability to convey very complex ideas.

Thirdly, I am grateful to my husband Jorge Castañón for his patience, encouragement and faith in me. Moreover, I would like to thank my family; especially my grandmother Dr. Silvia Medina, for being extremely involved in my life supporting every single one of my decisions, being a role model and always encouraging me to pursue my goals; as well as my parents Mr. and Mrs. Tony and Margarita Wood for their love, support and for always finding humorous ways to brag about me; my siblings Miriam, Yvonne and Tony Wood, for being my inspiration to keep going; and, my uncle Raul Robles for always making an effort to help me and my siblings.

I thought about listing all the names of people in the Department of Computational and Applied Mathematics (CAAM) who I would like to express my appreciation, but the list would be very long. As a consequence, I would like to thank the faculty, staff and students in the CAAM Department. Finally, I want to thank the

AGEP program and the NSF* for their continuous support throughout my graduate education.

* This work is made possible by the National Science Foundation Grant Number 0940902 and CMMI-1300477

Contents

Abstract	ii
Acknowledgments	iv
List of Illustrations	viii
List of Tables	x
1 Introduction	1
2 Basic Terminology and Background	7
2.1 The Clique Model	7
2.2 Clique Generalizations	8
3 The Minimal k-core Problem	11
3.1 Introduction	11
3.2 Formulation of the main problem, Basic Terminology and Background	16
3.2.1 The Cell Assembly: A Graph Theoretical Approach	17
3.2.2 k -assembly	19
3.3 Methods: Backtracking Algorithm Techniques	27
3.4 Numerical Results	36
3.5 Discussion	42
4 The Maximum Weighted Co-2-Plex Problem	45
4.1 Introduction	45
4.2 Basic Terminology and Background	46
4.3 A Tractable Instance of the Maximum Weighted Stable Set Problem	48

4.3.1	Minty's Algorithm	50
4.3.2	Tamura and Nakamura's Correction	56
4.4	Algorithms for Finding the Maximum Weighted Co-2-Plex in a {claw, bull}-Free Graph	60
4.4.1	The reduction	60
4.4.2	Generalization of Minty's Algorithm to Solve the MWC2P Problem in {claw, bull}-Free Graphs	65
4.4.3	Maximum Weighted White Augmenting Paths	69
4.4.4	Correctness of the Algorithm	70
4.5	Numerical Results	73
4.6	Discussion	77
5	The Minimum k-core Problem	78
5.1	Formulation of the Binary Integer Program	79
5.2	Linear Relaxation and Convex Hull	80
5.3	Branch and Bound	83
5.4	Valid Inequalities and Cutting Plane Algorithms	87
5.5	Branch and Cut	96
5.6	Numerical Results	99
6	Conclusion	110

Illustrations

1.1	A five vertex complete graph.	2
1.2	3-core	4
1.3	7-plex	4
3.1	Threshold function f_k for $k = 2$. On the left, we see the original graph with only the given set $S = \{1, 2, 6\}$ excited, $f_k(S)$ in the middle, and $f_k^2(S)$ in the right.	19
3.2	Cell assembly vs. k -assembly for $k = 3$. The graph on the left satisfies the definition of a cell assembly, but not of k -assembly. The graph on the right is a 3-assembly with $c(u, v) = 1$ for all $e = uv \in E$	21
3.3	Backtrack search tree for a given graph G . The root of the tree contains the entire graph and the leaves contain minimal k -cores or extensions that made the algorithm backtrack. The sets kcore, not and candidates follow the definitions of Algorithm 3.4	35
4.1	On the left, we see a claw and a bull on the right.	46
4.2	Example of a G, G' pair via the reduction.	61
4.3	A super free vertex of type I	66
4.4	A super free vertex of type 2	67
4.5	A free vertex of type I	67
4.6	A free vertex of type II	67
4.7	Bounded vertices of type I	68

4.8	A bounded vertex of type II	68
4.9	A bounded vertex of type III	68
4.10	A path, an Alternating Path and two White Alternating Paths	69
4.11	The Complement of a Claw and a Bull	73
4.12	A Butterfly and a Triangle are triangle-free graphs in the complement	75
5.1	Partial branch and bound tree for Problem 5.4	84
5.2	The node associated with S_1 is pruned by optimality	85
5.3	Partial branch and bound tree for Problem 5.4	86
5.4	The node associated with S_{21} is pruned by bound	86
5.5	Complete branch and bound tree for Problem 5.4	87
5.6	Partial branch and cut tree for Problem 5.4.	97
5.7	The node associated with S_1 is pruned by bound.	98
5.8	Complete branch and cut tree for Problem 5.4.	98
5.9	Perfomance of branch and bound and branch and cut to solve Problem 5.1 for $k = 2$	100
5.10	Perfomance of branch and cut to solve Problem 5.1 for $k = 2$	101
5.11	Number of subproblems solved by branch and bound and branch and cut for $k = 2$	102
5.12	Number of subproblems solved by branch and cut $k = 2$	103
5.13	Number of subproblems decrease when using branch and cut for $k = 2$.	104
5.14	Performance of branch and bound and branch and cut to solve Problem 5.1 for $k = 3$	105
5.15	Number of subproblems solved by branch and bound and branch and cut for $k = 3$	106
5.16	Number of subproblems solved by branch and cut for $k = 3$	107
5.17	Number of subproblems decrease when using branch and cut for $k = 3$.	108
5.18	Performance of branch and cut approach vs Matlab solver	109

Tables

3.1	Algorithm 3.4 performance for $n = 10$ and $p = 0.1, 0.5$ and 0.7	38
3.2	Algorithm 3.4 performance for $n = 15$ and $p = 0.1, 0.5$ and 0.7	39
3.3	Algorithm 3.4 performance for $n = 20$ and $p = 0.1, 0.5$ and 0.7	40
3.4	Algorithm 3.4 performance for $n = 25$ and $p = 0.1, 0.5$ and 0.7	41
3.5	Algorithm 3.4 performance for 5-regular graphs with $n = 30$ and $p = 0.1, 0.5$ and 0.7	42
4.1	Performance of Algorithm 4.4.2 and the Reduction Algorithm on the complement of Mycielski graphs	74
4.2	Performance of Algorithm 4.4.2 and the Reduction Algorithm on graphs generated with a triangle as base for replication	76
4.3	Performance of Algorithm 4.4.2 and the Reduction Algorithm on graphs generated with a butterfly as base for replication	76

Notation

$G = (V, E)$	The graph G with vertex set V and edges E
$cl(S)$	The closure of $S = (\tilde{V}, \tilde{E})$ where $S \subseteq G$
$c(u, v)$	The weight associated with $uv \in E$
$f_k^n(S)$	The n^{th} iteration of the threshold function f
C	A clique $C = (\tilde{V}, \tilde{E})$ where $C \subseteq G$
K	A k -core $K = (\tilde{V}, \tilde{E})$ where $K \subseteq G$
\tilde{K}	A k -plex $\tilde{K} = (\tilde{V}, \tilde{E})$ where $\tilde{K} \subseteq G$
$deg(v)$	The degree of a vertex $v \in V$
$\delta(G)$	The degree of a vertex of minimum degree
$N(v)$	The set of neighbors of a vertex $v \in V$
$*$	

*For more details on the notation see [5]

Chapter 1

Introduction

The study of social networks was first introduced in the context of sociology to study human interaction. A social network is naturally represented as a graph $G = (V, E)$ with vertex set V and edge set E . The vertices of the graph represent actors and the edges the relations between them. The existence of an edge between a pair of actors may represent friendship, neighborhood, affinity for the same sport, if they work together or any other type of relationship.

Social network analysis had been used for years to analyze social relationships and their psychological effect among members of a group. In the early 1930's, the Austrian psychiatrist Jacob Moreno and the American psychologist Hellen H. Jennings founded the field of sociometry [33]. Their research approach utilized sociograms to visually represent groups. The individuals of a group were represented by dots and their relations by lines. The sociometric approach had a surprising acceptance among sociologists and psychologists. Nevertheless, the success of sociometry was brief. There were not enough tools to analyze and find patterns in sociograms due to the fact that the field lacked mathematical rigor. As a result the development of social network analysis took a multidisciplinary approach. For a detailed discussion about the history of social network analysis see [41].

In the late 1940's, graph theory started to gain recognition among social scientists. Forsyth and Katz [19] were among the first ones to give a rigorous mathematical approach toward the study of social networks. Their approach brought back the interest in sociometry and set a trend among researchers in their field. Influenced by the novel approach, in 1949 Luce and Perry [27] introduced the clique model to analyze experimental data. They defined a clique as a subset of vertices in a graph that contained all possible edges. That is, a subgraph in which any two vertices are adjacent to each other. Figure 1.1 shows an example of a clique.

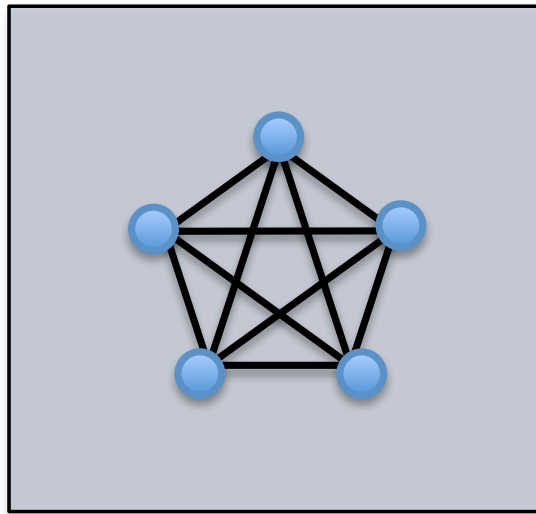


Figure 1.1 : A five vertex complete graph.

The introduction of the clique model received a large amount of attention in the field of sociology, due to its close relation with the foundational concept of cohesive subgroup. It was used to develop a non-rigorous approach towards the study of

network cohesion [18]. A cohesive subgroup consists of actors connected through dense relations that enable members to share information and perform as a group [53]. A clique induces a subgraph that is as dense as possible; hence, it is a perfect cohesive subgroup.

The structural properties of a clique are distance, diameter, domination, degree, density and connectivity. A clique can be defined in terms of its structural properties. The following definitions of a clique are equivalent: vertices are distance one away from each other; vertices induce a subgraph of diameter one; every one vertex forms a dominating set; each vertex neighbors all vertices; vertices induce a subgraph that has all possible edges; and all vertices must be removed to obtain a disconnected induced subgraph. For more details on this definitions, see [39].

The popularity of the clique model spread across multiple disciplines. For instance, in the field of biochemistry and genomics clique detection models have been utilized to model protein structure and drug discovery [9]. In neuroscience, cliques are utilized to study the visual cortex [30] and memory [35]. Chemists have used cliques to detect similarities among chemicals in databases [43]. In the area of national security, clique interdiction has been used to remove enemies [3] [56]. These are just a few areas that have taken advantage of the familiarity, reachability and robustness properties of the clique model. Unfortunately, the clique model is too restrictive for practical applications, leaves out any subgraph without all possible edges and is not robust to errors. Consequently, clique generalizations have been introduced to overcome the

disadvantages of the clique model.

Clique generalizations are subsets of the vertex set with structural properties similar to those of a clique. Yet, they violate or only ensure the presence of one elementary clique defining property. For instance, a k -plex is a subset of the vertex set in which each vertex neighbors all but k vertices. It is a clique generalization and it violates the degree and domination properties of a clique. Another example of clique generalization is a k -core whose definition only requires that each vertex has degree at least k . Figures 1.2 and 1.3 illustrate a 3-core and a 7-plex respectively.

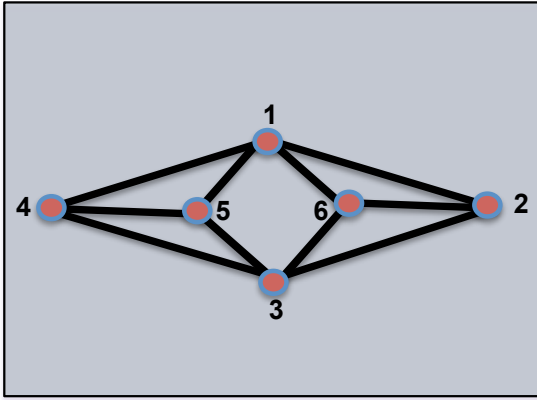


Figure 1.2 : 3-core

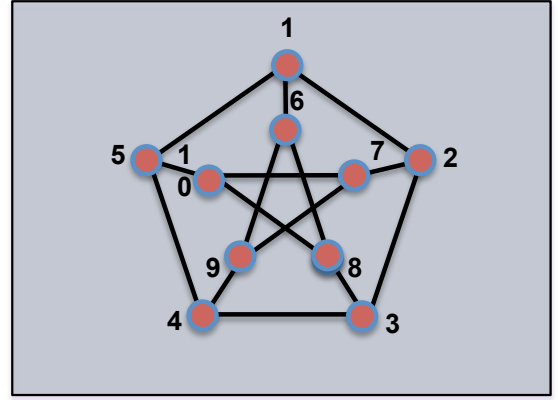


Figure 1.3 : 7-plex

This thesis presents a thorough study of the k -core polytope, the maximum weighted co-2-plex problem and the minimal k -core problem. These problems add to the current line of research on clique generalizations. They are useful on a wide range of applications, varying from sociological problems to biological models.

First, I discuss the minimal k -core problem, which is related to the study of associative memory. The concept of cell assembly was introduced by Hebb and formalized mathematically by Palm in the framework of graph theory. In the study of associative memory, a cell assembly is a group of neurons that are strongly connected and represent a “concept” of our knowledge. This group is wired in a specific manner such that only a fraction of its neurons will excite the entire assembly. There exist a link between the concept of cell assembly and the closure of a minimal k -core. I study a particular type of cell assembly called k -assembly. The goal of this particular project is to find all substructures within a network that must be excited in order to activate a k -assembly. Through numerical experiments, I confirm that these important subgroups overlap. To explore the problem, I present a backtracking algorithm to find all minimal k -cores of a given undirected graph. The problem of finding all minimal k -cores lies in the class of $\#P$ -complete problems. The proposed method is a modification of the Bron and Kerbosch algorithm for finding all cliques of an undirected graph. The results in the tested graphs offer insight in analyzing graph structure and help better understand how concepts are stored. The work presented about this problem has been published [55]. As a consequence, one may find similarities between this chapter and the published article.

Secondly, I introduce a study of the maximum weighted co-2-plex problem (MWC2P) problem for $\{\text{claw}, \text{bull}\}$ -free graphs. The MWC2P problem determines a subset of vertices of maximum total weight, in which each node has degree at most one.

The co-2-plex structure reveals the familiarity and reachability among its members. Therefore, this research applies to areas such as social network analysis and logistics. I present two polynomial time algorithms to solve the MWC2P problem for {claw, bull}-free graphs. The first algorithm reduces the original given graph to a claw-free graph and uses Minty's algorithm to solve the maximum weighted stable set problem. The second algorithm solves the problem directly on the given graph through a generalization of Minty's algorithm. The presented results add to the recent line of research focusing on stable set relaxations.

Finally, I present work on the minimum k -core problem. The minimum k -core problem asks for a k -core of minimum cardinality. The study of this problem started from the fact that a minimum k -core is a minimal k -core. In effort to understand k -assemblies the minimum k -core problem was model as a binary integer program [57]. In this work, I relax the integer program as a linear program and show that edge and cycle transversals of the graph give valid inequalities for the convex hull of k -cores.

Chapter 2

Basic Terminology and Background

This chapter introduces basic terminology to understand how k -cores and co- k -plexes relate to cliques, and why they are considered clique or stable set generalizations. For a detailed discussion on clique generalizations, see [2] and [50]. For an introduction to cliques, see [27].

2.1 The Clique Model

The study of network structure has been used for years to study social interaction. However, it was in 1949 when Luce and Perry [27] introduced the clique model to analyze experimental data.

Definition 2.1 Given a simple graph $G = (V, E)$ a subgraph $C = (\tilde{V}, \tilde{E})$ is a clique if for every two vertices u and $v \in \tilde{V}$ there exists an edge $e = uv \in \tilde{E}$.

Originally, the clique model was used to study social networks. Each vertex represented an actor and the edges represented the relations between them. In addition, this model was used to develop a non-rigorous approach towards the study of network cohesion [18]. A cohesive subgroup consists of actors connected through dense relations that enable members to share information and perform as a group [53]. Thus,

the notion of a cohesive subgroups is of high interest in social network analysis.

Definition 2.2 A cohesive subgroup is characterized by the following properties:

- (i) Mutuality of ties.*
- (ii) Closeness or reachability of subgroup members.*
- (iii) Frequency of ties among members.*
- (iv) High frequency of ties among members as opposed to those between members and non-members.*

A clique is the perfect example of a cohesive subgroup. Due to the fact that every pair of edges is adjacent to each other, the four properties of the cohesive subgroup definition are immediately satisfied. However, this definition is too restrictive and leaves out models that do not have all possible edges. As a consequence, clique generalizations have been introduced to overcome the limitations of the clique model.

2.2 Clique Generalizations

The clique model has different definitions in terms of distance, diameter, domination, degree, density and connectivity. The alternative definitions are fundamental in defining clique generalizations and must be introduced. For more details on this definitions see [39].

The following definitions are equivalent definitions of a clique:

1. **Distance:** Vertices are distance one away from each other.

2. **Diameter:** Vertices induce a subgraph of diameter one.
3. **Domination:** Every one vertex forms a dominating set .
4. **Degree:** Each vertex neighbors all vertices.
5. **Density:** Vertices induce a subgraph that has all possible edges.
6. **Connectivity:** Need to be remove all vertices to obtain a disconnected induced subgraph.

A clique generalization is characterized by at least one of the following properties:

- (i) it restricts a violation of an elementary clique-defining property.
- (ii) it ensures the presence of an elementary clique-defining property.

Throughout this thesis, I will refer to the concepts of maximum and minimum degree, distance and diameter as δ , Δ , d_G and $diam$ respectively. Some examples of clique generalizations are the following: a k -core induces a graph with $\delta(G[K]) \geq k$; a k -plex induces a graph with $\delta(G[K]) \geq |K| - k$; a k -clique has the property that $d_{G[K]}(u, v) \leq k \forall u, v \in K$ and a k -club induces a graph with $diam(G[K]) \leq k$.

A k -core is a clique generalization that ensures the presence of an elementary clique-defining property in terms of degree. A k -plex is an example of a clique generalization that violates an elementary clique defining property. It does not satisfy the definition of a clique in terms of degree and domination. The complement of a k -plex is a co- k -plex, and the complement of a clique is a stable set. A co- k -plex is a stable

set generalization. Now, I established the relation between cliques and co- k -plexes that will be studied later on this thesis. The problems presented on this thesis deal with the study of k -cores and co- k -plexes.

Chapter 3

The Minimal k -core Problem

3.1 Introduction

The brain's complex networks of neurons have been studied in an effort to understand human cognition and behavior. In parallel, graph theory and combinatorial optimization have focused in understanding the structure and dynamics of networks that arise from a wide spectrum of applications. In this work, I present mathematical techniques that provide insights in network structure. This is important to the study of the brain since it allows us to recognize structures that play key roles in certain fundamental mental processes. In particular, I focus on the relationship between the study of networks and memory.

Network structure and architecture has been studied to understand sociological and biological problems, mostly to identify cohesive subgroups within social and biological networks. The analysis of subgroups within a network serves to identify the most influential elements in a group; and to understand the interactions between members. Although brain networks are extremely complex, they share certain characteristics with social and biological networks. For further discussion, see [53], [6] and [42]. In particular, the study of interactions within a group is important to the study

of neuronal networks, since brain connectivity is crucial to process information. For a more detailed discussion about the relationship between networks and its applicability to the study of the brain see [44] and [48]. In this chapter, I study two specific network structures, namely a clique and a k -core, and its potential applications to the study of associative memory.

A clique is a subnetwork in which the actors are more tied to one another than to other members of the network. In terms of the brain, the actors are neurons and the ties between them represent synapses between these neurons. A clique can be seen as a group of neurons that collectively respond to a particular stimuli. The Hebbian theory of learning is often paraphrased as “Cells that fire together wire together” and refers to groups of neurons that fire in synchrony [21]. In other words, events that occur simultaneously are associated in memory. For instance, in a clique it is only necessary to give excitatory input to a fraction of the clique in order to make the entire network fire. In 1949 Luce and Perry introduced the clique model to analyze experimental data [27]. In addition, this model was used to develop a non-rigorous approach towards the study of network cohesion [18]. The clique model has gained popularity for being the perfect cohesive subgroup due to the existing relationship between each one of its members [53]. As a consequence, neural cliques have been used to model computation in the visual cortex [30], differential memory consolidation [35] and to understand episodic experiences in the hippocampus [25]. Nonetheless, the clique model has limitations and leaves out structures that still respond collectively to

certain stimuli if there is not a connection between each pair of neurons. Consequently, it is important to consider structures with properties similar to cliques, even if they are not maximally connected, such as the ones introduced by Seidman and Foster [47]. One of these structures is a k -core, which is a subgraph with minimum degree greater than or equal to k . For more details on models to overcome limitations of cliques see [2] and [50]. Throughout this work I focus on the relationship between k -cores and the insights they provide in the study of associative memory.

Memory is a fundamental mental process in the brain. Some of its attributes are to represent concepts and objects in the brain and recall information. In addition, memory is closely connected to the perceptual and learning processes. Donald Hebb in an effort to understand the behavior of the human brain introduced the term “cell assembly”. He defined it as a group of neurons that are strongly connected and represent a “concept” of our knowledge [21]. It refers to a memorized pattern in the auto-associative memory scheme, and according to Hebb’s definition it plays an important role in the structural change of long-term memory. For more details on associative memories as brain models and its storage capacities see [36]. The aforementioned definition can easily describe features of memory and its relations with other processes. Nevertheless, it is not known if the relations described by cell assemblies exist. If they were to be real, then the nodes of a given network could represent portions of a cell assembly, and its connections will describe the flow of activity in the cortex . For further discussion, see [12].

Hebb's definition of cell assembly created a gateway to research involving neuroscience and advanced mathematical techniques. Topology has been used to study stimulus reconstruction, and the used representation is close in spirit to Hebb's cell assembly [16]. Although the mathematical techniques utilized are different, stimulus reconstruction is related to the work presented in this paper since it helps to describe activity patterns of neuronal population during cognition. In addition, dynamical systems have been used to understand how knowledge and events are represented and processed in the brain [51]. This type of work studies the dynamics of cell assemblies and gives mathematical expression of the hypothetical dynamics of neuronal populations in the cortex.

Until today, there does not exist enough evidence to contradict Hebb's definition of cell assembly. From the physiological point of view, the idea requires variable excitatory synapses that obey Hebb's rule. In other words, the connectivity is enhanced by coincident pre- and postsynaptic activity [37]. However, this specific point of view is difficult to test due to the unavailability of experimental data. Valentino Braitenberg was the first one to give interpretation to the theory of cell assemblies in terms of neuroanatomy and neurophysiology [7]. Most of the ideas presented on Braitenberg's work have been thoroughly explored and served as the basis of cell assembly theory. For a detailed discussion on the current state of cell assembly theory see [38]. According to Hebb's definition, a cell assembly represents only one concept in our brain. This implies that there must exist a large number of cell assemblies in order to

store all the concepts in the brain, and it is still not possible to identify all of them. For an efficient and reliable statistical method to detect and identify members of an active cell assembly directly as significant spike synchrony patterns see [40]. In an effort to investigate if the cortical network is sufficient to contain all of our concepts Palm formulated the main problem of the theory of cell assemblies. The problem asked for the total number of assemblies of a given network. In theory, it is possible to find all cell assemblies and determine the solution to the problem. However, due to the complexity of the definition; the number of neurons on a brain-sized neuronal network; and the number of connections per neuron, it still may not be possible, in practice, to solve the problem of finding all cell assemblies. Therefore, let us focus on a particular type of cell assembly called a k -assembly.

In this work, I extend Palm’s graph theoretical approach towards understanding memory. I show a connection between the concept of a cell assembly and the definition of a k -core, which allowed us to define a k -assembly. I go beyond Palm’s main problem of the theory of cell assemblies that asks for the total number of assemblies at a fixed threshold, to ask for all the substructures whose excitation cause the activation of an entire assembly for a given threshold. I solve the aforementioned problem by finding all minimal k -cores of a given undirected graph via a backtracking algorithm. I present complexity results related to k -cores to highlight the mathematical difficulty of the problem and provide numerical results to validate the proposed algorithm.

The following section provides the necessary background to understand the math-

emational definition of cell assembly and k -assembly as well as a brief overview of backtracking algorithms. In particular, I discuss the Bron and Kerbosch algorithm whose backtracking structure is the essence of the algorithm proposed to solve the desired problem. The proposed algorithm to find all minimal k -cores and its complexity are discussed in the methods section followed by numerical results. Lastly, a discussion of the work introduced in this paper is given.

3.2 Formulation of the main problem, Basic Terminology and Background

The goal of this project is to find all substructures within a graph $G = (V, E)$ that must be excited in order to activate a particular type of cell assembly that will be defined in this section, the k -assembly. In the graph G , each vertex v in the vertex set V represents a neuron, and each edge e in the edge set E represents a connection between two neurons, the *threshold* is denoted as the minimum number of inputs each node receives in order to become excited. Throughout this paper, the threshold value will be fixed to a particular given integer k . However, it is of high interest to study the behavior of networks as the value of k changes with respect to time.

In this section, the reader will be introduced to basic terminology necessary to link the concepts of cell assembly and k -assembly. The purpose of this section is to state definitions that will be referred throughout this article. For a detailed discussion of cell assemblies see [21].

3.2.1 The Cell Assembly: A Graph Theoretical Approach

In 1981, Palm proposed a mathematical interpretation of Hebbian theory in the framework of graph theory. He gave a mathematical interpretation to the cell assembly. In order to understand Palm's mathematical definition of a cell assembly, the reader must be introduced to some background definitions.

Given a simple graph $G = (V, E)$ in which each vertex v in the vertex set V represents a neuron, and each edge e in the edge set E represents a connection between two neurons, the *threshold* is denoted as the minimum number of inputs each node receives in order to become excited. Throughout this paper, the threshold value will be fixed to a particular given integer k . However, it is of high interest to study the behavior of networks as the value of k changes with respect to time.

Given a weighted graph (G, c) , where the weight $c(u, v)$ represents the strength of the synapses from neuron u to neuron v for all edges $uv \in E$. For the rest of this paper, I fix the value of $c(u, v) = 1 \forall uv \in E$.

Definition 3.1 Given $S \subseteq V$ and an integer k , a threshold function f_k is described by

$$f_k(S) = \{v \in V \mid \sum_{u \in S} c(u, v) \geq k\}$$

The resulting active set of nodes of $S \subseteq V$ at a threshold k is obtained when S is given as an input to the threshold function f_k . That is, given a subset S of activated nodes, other nodes in the graph will become activated if they satisfy the threshold

inequality, for simplicity I denote $f_k^i(S) = f_k(f_k^{i-1}(S))$ for $i \geq 2$ and $f_k^1 = f_k$. Figure 3.1 illustrates this process for $k = 2$.

Definition 3.2 A subset of vertices S is called invariant if $f_k(S) = S$.

Definition 3.3 The closure of S , denoted $cl_k(S)$, is the invariant set generated when $f_k^n(S) = f_k^{n-1}(S)$ for some $n \geq 1$.

In Figure 3.1, the closure of the set $S = \{1, 2, 6\}$ is achieved when $n = 3$, and it is the entire vertex set V .

Definition 3.4 A subset S is called persistent if $f_k(S) \supseteq S$, and it is called minimal persistent if no proper subset of it is persistent.

In Figure 3.1, the set $S' = \{1, 2, 3, 6\}$ is persistent when $k = 2$. However, $S = \{1, 2, 6\}$ is a persistent subset of S' , which implies S' is not minimal.

Definition 3.5 A subset S is called weak if there exist an $n \geq 1$ such that $f_k^n(S) = \emptyset$.

In Figure 3.1, the set $S' = \{1, 2\}$ is weak, since $f_k(S') = \{6\}$ and $f_k^2(S') = \emptyset$.

Definition 3.6 A tight set is a persistent set P in which every persistent subset of P whose complement in P is not weak and excites the whole of P .

Finally, the reader has the necessary background concepts to understand Palm's mathematical definition of *cell assembly*.

Definition 3.7 A cell assembly (at a threshold k) is the closure of the tight set.

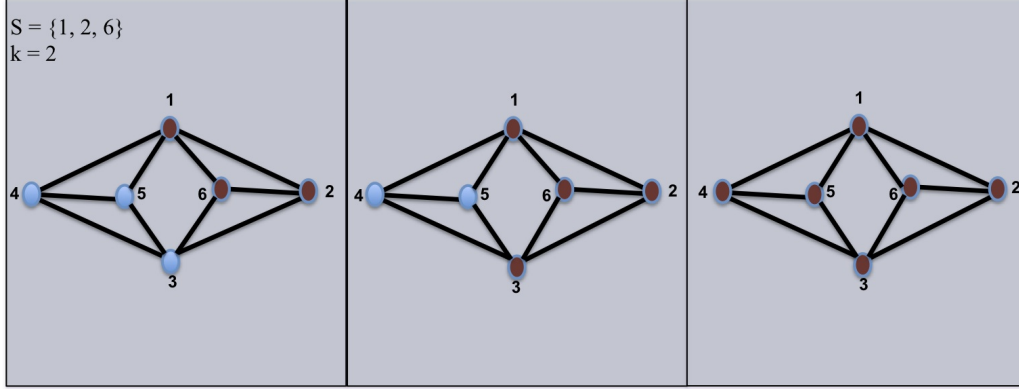


Figure 3.1 : Threshold function f_k for $k = 2$. On the left, we see the original graph with only the given set $S = \{1, 2, 6\}$ excited, $f_k(S)$ in the middle, and $f_k^2(S)$ in the right.

The mathematical definition of cell assembly encompasses a variety of tight sets. For instance, in Figure 3.1, S is a tight set and any superset of S is also a tight set. Yet, Palm proposed that a minimal persistent set is a tight set [37]. Therefore, I focus on the study of cell assemblies generated by minimal persistent sets.

3.2.2 k -assembly

Seidman introduced k -cores to study network structure, and demonstrate that k -core cohesion increases as k increases [46]. He defined a k -core as a maximal connected induced subgraph with degree greater than or equal to k . The maximal property of Seidman's definition will not be considered for the topic presented in this paper. In other words, I define a k -core to only be a subgraph with minimum degree at least k .

Definition 3.8 A subgraph $K \subseteq G$ is a k -core if $|N(v) \cap V(K)| \geq k \forall v \in V(K)$.

Definition 3.9 A k -core is minimal if no proper subset of its vertices induces a k -core.

It is clear by the definition that the subgraph generated by $f_k(\tilde{V})$, for some $\tilde{V} \subseteq V$ is a k -core if and only if \tilde{V} is a persistent set. That is if $f_k(\tilde{V})$ is a k -core, then for all $\tilde{v} \in \tilde{V} \mid |N(\tilde{v}) \cap f_k(\tilde{V})| \geq k$, which implies $\tilde{V} \subseteq f_k(\tilde{V})$. Likewise, if \tilde{V} is a persistent set, then $\tilde{V} \subseteq f_k(\tilde{V})$, which implies $f_k(\tilde{V})$ is a k -core. In addition, note that for an unweighted graph, the threshold function definition of a tight set S becomes $f_k(S) = \{v \in V \mid |N(v) \cap S| \geq k\}$, that is $cl(S)$ generates a k -core. By definition, a k -core is tight as long as its complement is not weak, since every subset of its vertex set is persistent. Hence, the closure of any k -core generates a cell assembly.

The definition of cell assembly tell us that it only takes a fraction of the assembly to get excited in order to excite the entire assembly. However, the motivation and focus of my work comes from the study of cell assemblies generated by tight sets that are minimal, that is the deletion of any node from the set generates a subset that is not tight. In addition, the mathematical definition of cell assembly for its study on simple graphs follows the definition of a k -core. According to Palm's definition of tight set, a particular type of tight set is a minimal k -core. Hence, the vertex set of a minimal k -core generates a particular type of cell assembly called k -assembly.

Definition 3.10 A k -assembly is the closure of a minimal k -core.

Recall this definition only holds for cases in which the G has $c(u, v) = 1$ for all $e = uv \in E$. In figure 3.2, we observe on the left that any two adjacent vertices satisfy the definition of cell assembly for $k = 3$, since the edges have weights with

value greater than one. Nevertheless, a set with less than $k + 1$ vertices cannot be a minimal k -core, and its closure is not a k -assembly. In contrast, the graph on the right has every edge with weight equal to one, and the entire vertex set constitutes a 3-assembly.

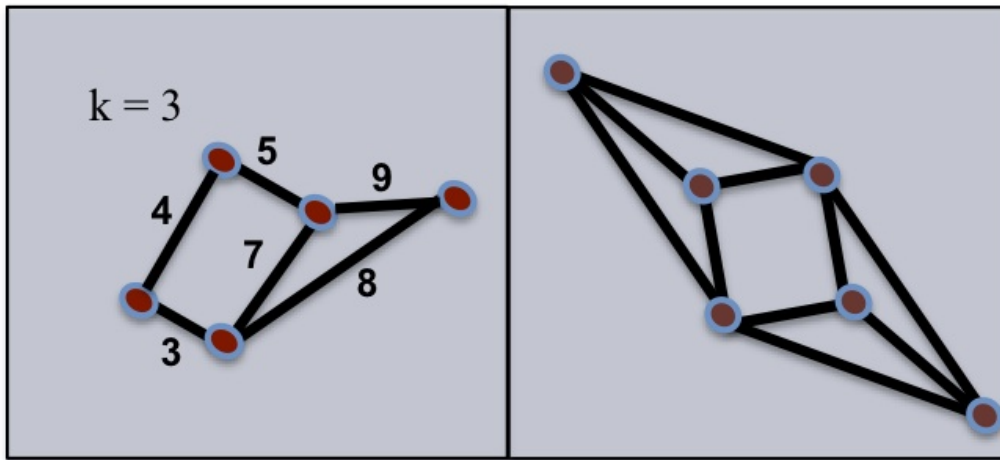


Figure 3.2 : Cell assembly vs. k -assembly for $k = 3$. The graph on the left satisfies the definition of a cell assembly, but not of k -assembly. The graph on the right is a 3-assembly with $c(u, v) = 1$ for all $e = uv \in E$

The definition of k -assembly and cell assembly served as motivation to solve the problem of finding all cell assemblies and tight sets that generate them, in particular minimal k -cores. The remainder of this paper focuses on solving the problem of finding all minimal k -cores for a given simple undirected graph. Nevertheless, finding k -cores is not an easy task, and I briefly discuss some complexity properties of problems that deal with k -cores in the rest of this section.

Theorem 3.1 The k -core containment problem is NP-complete

Proof of Theorem 3.1. The decision version of the problem is the following:

Instance: Given a graph G and integers $s \leq |V|$

Question: Does G have a k -core of size s ?

Clearly, the k -core problem belongs to NP since given a solution of the problem, a nondeterministic Turing Machine checks if the choice is true in polynomial time. Furthermore, If we restrict the k -core problem by considering only instances in which the cardinality of the k -core $s = k + 1$, then we get the clique problem [20]. Hence, the k -core containment problem is NP -complete. \square

The problem of finding all minimal k -cores also requires graphical enumeration which refers to the art of counting the number of graphs with a specific property. Note that for some problems to count the number of graphs with a given property is harder than to determine if there exists a graph that satisfies such a property. For instance, “Given a graph G and a fixed value $k > 0$, how many distinct k -cores are there for G ? ” is not a trivial problem and it empirically depends on the density of the graph. Enumeration problems associated with NP -complete problems are NP -hard [20]. This is true since the enumeration version of the problem must be at least as hard as the decision version of the problem. Hence the enumeration of k -cores is NP -hard.

To study in depth enumeration problems the class $\#P$ was introduced [52].

Definition 3.11 The class $\#P$ contains all problems computed by nondeterministic polynomial time Turing machines that have the additional facility of outputting the

number of accepting computations.

Moreover, $\#P$ -complete is the analog definition of NP -complete for P . The class $\#P$ asks for the number of solutions rather than its existence. For NP -complete problems counting the number of solutions is $\#P$ -complete. Therefore enumeration of k -cores belongs to the class of $\#P$ -complete problems.

The detection of minimal k -cores is important since they denote the structural motifs (i.e. building blocks of more complex networks) that must be excited in order to propagate the excitation in the graph. The idea of k -assembly is related to motif detection [49]. However, instead of restricting it to the study of motifs of certain size, it focuses on the study of subgraphs that pass a certain threshold.

Previous Work on Solving the Minimal k -core Enumeration Problem

The fact that a clique with vertex set cardinality $k + 1$ is a minimal k -core allows us to say that algorithms performing clique enumeration were the first ones to attack a subset of the problem I present in this paper.

The Maximal Clique Enumeration Problem (MCEP) asks to compile a list of all maximal cliques in a given undirected graph G . Besides its applications in sociological problems, it is also useful in the study of biological networks [10]. MCEP in the worst case scenario runs exponential with respect to the number of vertices. More specifically, the maximum number of maximal cliques in an n vertex graph is $3^{\frac{n}{3}}$ [32].

In other words, it has been proved that there may be a graph with an exponential number of maximal cliques, which implies that any algorithm that solves MCEP for an arbitrary given graph would be exponential.

Bron and Kerbosch (B&K) developed a backtracking algorithm to solve MCEP in 1973 [8]. Although other algorithms to solve the problem were developed around the same period [1], the B&K approach is still one of the most widely known to solve this problem and it is used as a basis for other algorithms that solve MCEP. For further discussion on modifications of B&K, see [11]. The B&K algorithm depends on the number of nodes in the graph, and numerical experiments show it runs in $O(3.14^{\frac{n}{3}})$ on Moon-Mooser graphs with a theoretical limit of $3^{\frac{n}{3}}$. The B&K Algorithm will be discussed in more detail in the following section.

As MCEP, the Minimal k -core Enumeration Problem (MKEP) asks to create a list of all minimal k -cores in a given undirected graph. There is not a known bound for the maximum number of minimal k -cores on a given graph. However, the fact that a clique with vertex set cardinality $k + 1$ is a minimal k -core, intuitively tells us that the number of minimal k -cores grows exponentially in the worst case scenario.

A solution to MKEP through exhaustive search has been proposed, it follows the structure of a branching algorithm [15]. Their algorithm, as well as the one I propose in the methods section initially obtain the maximum k -core. The following greedy algorithm obtains the maximum k -core in polynomial time [2]:

Algorithm 3.1 [Maximum k -core]

MaximumKcore(G)

if G is empty

0. **End**

else

1. Choose a vertex v

of minimum degree $\delta(v)$

if $\delta(v) \geq k$

2. The minimum k -core is found

3. **End**

if $\delta(v) < k$

4. MaximumKcore($G := G \setminus v$)

A description of the algorithm proposed by [15] is the following:

Algorithm 3.2 [k -core Enumeration of G]

Given an undirected graph $G = (V, E)$

0. **if** G is a minimal k -core

End

else

1. Find the maximum k -core, call it H

for each $v \in V(H)$

2. $V(G) := V(H) \setminus v$

3. Go to step 1

The algorithm described above finds all minimal k -cores of a given graph. However, a major disadvantage is the fact that it may return the same minimal k -core multiple times. It initially checks if the given graph G is a minimal k -core, and stops in the case it is in fact a minimal k -core. Otherwise, it proceeds to find the maximum k -core, and then minimal k -cores. No numerical results are given for the k -core enumeration approach performance. Yet, it is mentioned that it takes minutes to enumerate the k -cores of a graph with a vertex set of 10 nodes. The algorithm I developed to solve MKEP will be discussed in the methods section and its performance is analyzed in the numerical results section.

3.3 Methods: Backtracking Algorithm Techniques

Backtracking is a type of recursive strategy commonly used to find all the solutions of some problem. It incrementally builds a tree in a such a way that it faces a number of options at each level, and tries all of them. In a problem with N possible solutions, exhaustive search techniques evaluate all the options in N trials. In contrast, a backtracking algorithm yields the solution with less than N trials, and its solution space is organized as a tree. Initially, it starts at the root of the tree and proceeds to make a choice between one of its children, then it continues to make a choice among the children of each node until it reaches a leaf. Each leaf is either a solution of the problem or does not lead to a solution, and at that point the algorithm backtracks. For more details on backtracking algorithms see [23] and [24].

In the remaining of this section, I discuss two backtracking algorithms. The first one solves the problem of finding all maximal cliques in a given graph. The second one offers a solution to the problem of listing all minimal k -cores of a graph. In addition, an example of a backtracking tree is shown to illustrate the second presented algorithm.

The Bron and Kerbosch Algorithm for Finding All Cliques of an Undirected Graph

The B&K algorithm utilizes a recursively defined extension operator that is applied to three sets: *compsub*, *not* and *candidates*. The set *compsub* contains the nodes

already defined as part of the clique and it is initially empty. The set *candidates* is the set of nodes adjacent to all nodes in the set *compsub*. The set *not* stores the nodes that had already been processed, leading to a valid extension of the set *compsub* and should remain ignored. In addition to these three sets, there are nodes that are not considered at each step.

In order to obtain all maximal cliques, a backtrack search tree is constructed through recursive calls to the extension operator. Every time the recursion is called the three main sets are modified. The set *not* and *candidates* are given to the extension operator as input parameters and are locally defined. In contrast, the set *compsub* is globally defined and behaves like a stack. It is important to point out, that if at some point the set *not* contains a vertex that is adjacent to all vertices in *compsub*, then the algorithm backtracks since no further selection of candidates will lead to obtaining a maximal clique from the current configuration of the set *compsub*. The basic mechanism can be described in the following pseudocode:

Algorithm 3.3 [Bron and Kerbosch]

Extension(*compsub*, *candidates*, *not*)

if *candidates* = \emptyset and *not* = \emptyset

1. Report *compsub* as a maximal clique

else For each vertex $v \in \textit{candidates}$:

2. Select a candidate s

3. Add s to *compsub* such that

new *compsub* := *compsub* \cup s

4. Create new sets *candidates* and *not*

by removing all points not connected

to s and store old sets, that is

candidates := *candidates* $\cap N(s)$

not := *not* $\cup N(s)$

5. Extension(*compsub*, *candidates*, *not*)

6. Upon return, remove s from *compsub*

and add it to *not*

compsub := *compsub* $\setminus s$

not := *not* $\cup s$

End

A clique is found if and only if the sets *candidates* and *not* are empty. If *not* is not empty then the current configuration of the set *compsub* is not maximal. The algorithm terminates if there is no candidates left or if there is an element in *not* that is connected to all elements in the set *candidates*. If the second condition for termination is met, then the addition of any candidate to *compsub* will not be maximal.

To optimize the algorithm and make it terminate as early as possible, the number of times the extension operator is called must be minimized. To do this, every node in *not* is assigned a counter that indicates to how many candidates a node is not adjacent (or disconnected). I then proceed to pick the node with the smallest number of disconnections and on each step select a candidates not adjacent to this node.

Algorithm for Finding All Minimal k -cores of an Undirected Graph

The problem of finding all minimal k -cores of a given graph is computationally expensive. There exists a variety of algorithms to find all cliques in a given undirected graph. However, the B&K algorithm is commonly used to find all maximal cliques, since numerical experiments support its efficiency. I propose a modification of the B&K algorithm to find all minimal k -cores on a given graph.

As in B&K, the algorithm presents a backtracking technique to find all minimal k -cores. Three sets are utilized to obtain all minimal k -cores recursively, namely *kcore*, *not* and *candidates*. However, since a k -core is a generalization of a clique,

and every clique on $k + 1$ or more nodes contains a minimal k -core, but not every minimal k -core is a clique, there are some subtle changes in the definition of the sets. For example, I take into account that given a connected simple undirected graph, all minimal k -cores must be contained in the maximum k -core, which can be found in polynomial time (for more details see [2]).

The set *kcore* stores the nodes that are part of a k -core and is initially the entire vertex set. The set *candidates* contains the nodes that can be deleted to obtain a minimal k -core. The set *not* represents the nodes that had already been processed and cannot be deleted from *kcore*. As in B&K, these three sets are modified by a recursively defined extension operator. The set *kcore* is globally defined, whereas, the sets *not* and *candidates* are locally defined and handed as parameters to the extension operator.

We construct our backtracking search tree by recursively calling the extension operator. At the root of the search tree, the number of branches generated is equal to the cardinality of the set *candidates*. Each branch corresponds to removing one vertex from the configuration of the set *kcore*, and creating new sets *candidates* and *not*. The algorithm always selects the vertex of smallest degree one at a time. It continues traversing the search tree on a depth first search approach if there is at least one vertex in the set *candidates* whose deletion leads to obtaining a k -core of smaller cardinality and backtracks if the configuration of *kcore* cannot lead to returning a minimal k -core. That is, if the set *not* contains vertices that must be deleted in order

to obtain a minimal k -core then no further calls to the extension operator will lead to a valid configuration of the set $kcore$. Hence, such a branch must not be extended.

The basic idea behind the algorithm is the following:

Algorithm 3.4

0. Obtain maximum k -core

Extension ($kcore$, not , $candidates$)

if $candidates = \emptyset$ and $kcore \setminus v_i$

does not induce a k -core $\forall v_i \in not$

1. Report $kcore$ as a minimal k -core

else For each vertex $v \in candidates$:

2. Select a candidate s of smallest degree

3. Remove s from $kcore$ such that

$kcore := kcore \setminus s$ and

$candidates := candidates \setminus s$

4. Create new sets $candidates$ and not

and store old sets, that is

$candidates := \text{set of all candidates}$

$v \in V \setminus not$ that still leave a k -core

5. Extension($kcore$, not , $candidates$)

6. $not := not \cup s$

$kcore := kcore \cup s$

$candidates := candidates \setminus s$

The majority of the steps described above are straight forward to implement. However, there are several different options on how to implement step 2, which is how to select a well chosen candidate to minimize the number of times the extension operator is called. At the moment, it is impossible to give a good theoretical explanation on why one way to choose a candidate is better than other. They vary on a case by case basis, and its efficiency is determined by observations on numerical experiments. I chose to select a candidate of minimum degree because this way ensures that the set *not* is filled in correctly. However, modifying the original given set of *candidates* to be in the form required to be a candidate and selecting the candidate of maximum degree will also yield a solution to the problem. Figure 3.3 displays the backtrack search tree of the algorithm for a given graph G .

Now, I have to show that the proposed algorithm terminates and performs correctly. Clearly, for a given graph G with finite vertex and edge sets the algorithm terminates since the number of subgraphs to enumerate is finite. However, the number of subgraphs in a given graph G depends on its structure, and it may be very large for dense graphs. The next theorem is of extreme importance in showing correctness of Algorithm 3.4, since it guarantees that for every subgraph that contains a k -core all its minimal k -cores are generated without duplication.

Theorem 3.2 The extension of the backtracking search tree for a given configuration of the set $kcore$ by applying the extension operator generates all minimal k -cores without repetition that contain $kcore \setminus v_i \ \forall \ v_i \in candidates$

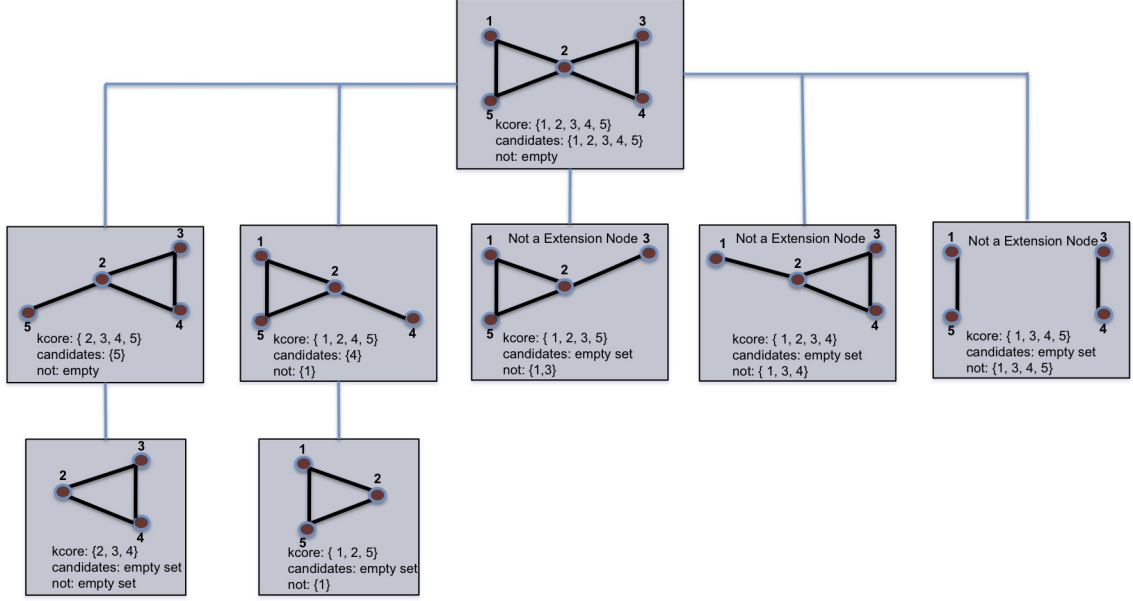


Figure 3.3 : Backtrack search tree for a given graph G . The root of the tree contains the entire graph and the leaves contain minimal k -cores or extensions that made the algorithm backtrack. The sets $kcore$, not and $candidates$ follow the definitions of Algorithm 3.4

Proof of Theorem 3.2. This proof is by strong induction on the cardinality of the set $kcore$.

For our base case, We consider $|kcore| = k + 1 \forall k > 0$. If $|candidates| = 0$ then k -core is minimal. Since we start with the largest k -core of the graph and our algorithm only allows to remove a vertex $v \in V \setminus not$ such that the subgraph obtain by the deletion of this vertex contains a k -core. The case $|candidates| > 0$ is not possible, since it implies that there exist a k -core of cardinality less than or equal to k which is false by definition of a k -core.

Now suppose that the statement is true for all $l > k \in \mathbb{Z}$ such that $l \leq N$, and that all minimal k -cores obtained by removing an element of not from the current

configuration of the set $kcore$ have been previously generated. We can suppose the later since it is guaranteed by our definition of the set $candidates$.

Consider a configuration of $kcore$ with cardinality $N + 1$. Let $\{v_1 \dots v_c\}$ represent the set of candidates for $c \geq 0$. If $|candidates \setminus v_i| = 0$ for some $0 \leq i \leq c$ then we have that $kcore$ is a minimal k -core.

If $|candidates \setminus v_i| > 0$ then we have the following two cases:

Choose \tilde{v} as in step 2 of our algorithm and create a new set of $candidates := candidates \setminus \tilde{v}$, proceed to call $extension(kcore \setminus \tilde{v}, candidates, not)$. If the cardinality of the new set of candidates is greater than 0, then by inductive hypothesis we have that the statement is true for $l = N + 1$. If it is zero then $kcore \setminus \tilde{v}$ is not minimal, and \tilde{v} is added to not . Which completes our proof and we get that Theorem 3.2 is true $\forall n \in \mathbb{Z}$. □

Since Theorem 3.2 is true for any subgraph of any given finite cardinality, we get that Algorithm 3.4 finds all minimal k -cores of a given undirected graph without repetition. In the following section, the results from running the backtracking algorithm for several test instances are presented.

3.4 Numerical Results

Algorithm 3.4 was implemented using C++ and tested in workstation with a AMD Opteron(tm) Processor 148. Results of numerical experiments are run to test the backtracking algorithm on random graphs. More specifically, I utilized graphs that

follow a Bernoulli process in the generation of edges, and are known as Bernoulli random graphs, as well as regular graphs. The existence of an edge in Bernoulli random graphs occur independently between each pair of nodes. For instance, given some probability p and the number of vertices n . There exists an edge (i, j) , where $i \neq j$ and $0 < i, j \leq n$. In contrast, regular graphs have the property that each vertex has the same degree.

A summary of the obtained results is presented at the end of this section, where one hundred Bernoulli random graphs were generated for each test instance, then the average time and number of k -cores were computed among the number of graphs that in fact contained at least one k -core for $k = 2, 3$ and 5 .

The average number of minimal k -cores is displayed to highlight the fact that the number of minimal k -cores depends on the density of the graph, and not on the value of k . Even though every k -core is a $k - 1$ -core, the fact that I restrict the solution set to k -cores that are minimal give us cases in which the number of k -cores is greater than the number of $k - 1$ -cores.

The tables below illustrate the performance of Algorithm 3.4 when the number of vertices $n = 10, 15, 20$ and 25 , the probability for generating an edge $p = 0.1, 0.5$ and 0.7 and the value of $k = 2, 3$ and 5 . # of k -cores denotes the average number of k -cores on the tested graphs, and # of Graphs is the number of graphs that at least contained one k -core.

In table 3.1, we observed the results for graphs with 10 vertices. The graphs

k	p	# of k-cores	Average Time	# of Graphs
2	0.1	1.1764	≈ 0 s	17
3	0.1	0	≈ 0 s	0
5	0.1	0	≈ 0 s	0
2	0.5	27.72	≈ 0 s	100
3	0.5	12.2041	≈ 0 s	98
5	0.5	1	≈ 0 s	6
2	0.7	57.14	0.0001 s	100
3	0.7	54.02	≈ 0 s	100
5	0.7	5.4634	≈ 0 s	82

Table 3.1 : Algorithm 3.4 performance for $n = 10$ and $p = 0.1, 0.5$ and 0.7

generated with probability 0.1 only had a few 2-cores, since they are not dense enough to even contain k -cores for larger values of k . As the probability increased, we observed that more minimal 3-cores and 5-cores were part of the random graphs. However, the average number of minimal 2-cores is always greater than 3-cores and 5-cores. It is important to point out, that we observe this behavior only because the vertex set cardinality is small. But, it is not always the case to have more minimal 2-cores than minimal 3-cores as we will see in later results.

In Table 3.2, the results for graphs with 15 vertices are displayed. We still observe a low existence of minimal k -cores for sparse graphs with $p = 0.1$. However, graphs

k	p	# of k-cores	Average Time	# of Graphs
2	0.1	1.9108	≈ 0 s	56
3	0.1	0	0 s	0
5	0.1	0	0 s	0
2	0.5	166.54	0.0636 s	100
3	0.5	303.01	0.059 s	100
5	0.5	25.22	0.0029 s	90
2	0.7	258.46	0.0577 s	100
3	0.7	630.02	0.0604 s	100
5	0.7	619.09	0.0457 s	100

Table 3.2 : Algorithm 3.4 performance for $n = 15$ and $p = 0.1, 0.5$ and 0.7

generated with probabilities 0.5 and 0.7 show a different behavior and contain a larger number of k -cores. Note that in contrast to graphs on 10 vertices, on these cases the number of minimal 3-cores is larger than the number of minimal 2-cores and decreases again for the number of 5-cores.

Table 3.3 displays the results obtained for random graphs with 20 vertices. In the set of graphs generated with $p = 0.7$, we observe that the average number of minimal 5-cores exceeds the average number of minimal 3-cores and 2-cores. The same behavior is observed in Table 3.4 for random graphs on 25 vertices with $p = 0.5$ and 0.7. In terms of k -assemblies, we observe that at a fixed threshold the number

k	p	# of k-cores	Average Time	# of Graphs
2	0.1	6.8370	1.8583 s	92
3	0.1	2	0.485 s	2
5	0.1	0	0 s	0
2	0.5	635.11	2.3256 s	100
3	0.5	3511.19	1.4819 s	100
5	0.5	2661.11	0.0029 s	100
2	0.7	791.66	2.0905 s	100
3	0.7	3902.45	2.3057 s	100
5	0.7	17010.33	2.9131 s	100

Table 3.3 : Algorithm 3.4 performance for $n = 20$ and $p = 0.1, 0.5$ and 0.7

of minimal sets that generate a k -assembly increase as k increases. This tells us that the number of minimal 2-cores is smaller than the number of minimal 3-cores and 5-cores, which is not true in general if the k -cores are not minimal.

In Table 3.4, we observe an interesting phenomena, which is that Algorithm 3.4 finds all minimal k -cores of a random graph faster when the graph is dense for the three values of k utilized to test it. Although, this result may seem counterintuitive, observations showed that the algorithm backtracks faster whenever it is dealing with a dense graph. Algorithm 3.4 initially takes longer to output the first minimal k -core for a dense graph than for a sparse one. However, after the first minimal k -core is

k	p	# of k-cores	Average Time	# of Graphs
2	0.1	25.13	122.0613 s	99
3	0.1	1.833	2.905	6
5	0.1	0	0 s	0
2	0.5	1990.34	85.1718 s	100
3	0.5	25318.58	101.519 s	100
5	0.5	84110.96	117.7972 s	90
2	0.7	1900.64	80.9009 s	100
3	0.7	16796.83	83.4447 s	100
5	0.7	211859.96	109.2354 s	100

Table 3.4 : Algorithm 3.4 performance for $n = 25$ and $p = 0.1, 0.5$ and 0.7

found; it backtracks to deal with more cases in which minimal k -cores in fact exist and with less configurations of the set *compsub* that do not lead to obtaining a minimal k -core.

In addition to Bernoulli random graphs, random 5-regular graphs with $n = 30$ were tested to check if we observe the same behavior as in random graphs, see Table 3.5. As expected they only had one minimal 5-core. However, they also contain a greater number of 3-cores than 2-cores.

The results for the 5-regular graphs are very similar regardless of the probability of their generation. This is due to the fact that they share the same structure.

k	p	# of k-cores	Average Time	# of Graphs
2	0.1	29229.97	2512.11 s	100
3	0.1	31860.98	398.71 s	100
5	0.1	1	≈ 0 s	100
2	0.5	29302.06	2512.46 s	100
3	0.5	31907.16	402.9 s	100
5	0.5	1	≈ 0 s	100
2	0.7	29217.7	2511.61 s	100
3	0.7	32036.35	392.18 s	100
5	0.7	1	≈ 0 s	100

Table 3.5 : Algorithm 3.4 performance for 5-regular graphs with $n = 30$ and $p = 0.1, 0.5$ and 0.7

Nonetheless, it is still necessary to check if these type of graphs follow the same behavior as Bernoulli random graphs, since the brain is neither completely random nor regular.

3.5 Discussion

In this chapter, I proposed a backtracking algorithm to find all minimal k -cores whose excitation can activate a k -assembly. The motivation to study this problem emerges from the urge to understand memory. Palm formulated the main problem of the theory of cell assemblies by asking the total number of cell assemblies at a given

threshold k . The proposed algorithm is closely related to this problem since it allows us to find the total number of subsets that generate k -assemblies on a given graph. Through numerical experiments I confirm that fractions of these important subsets overlap. These overlappings tell us that concepts are organized in groups and certain triggers activate associated memories.

An extension to the graph theoretical approach for the analysis of associative memory introduced by Palm is presented along with details on the derivation of the k -assembly from the cell assembly model. Although Algorithm 3.4 is not fast enough to solve the problem in a brain-sized neuronal network, it does offer a solution to the problem, permits us to analyze the structure of a given random graph and gain insight on understanding k -assemblies and cell assemblies. For instance, the fact that for some graphs there may be a larger number of minimal 5-cores than 3-cores allowed us to observe how the structures overlap. If we look at it in terms of memory, we can tell that certain nodes are members of several k -assemblies, and the absence of one of them may change the structure of the network completely. If larger data sets become available we could use standard techniques for network clustering or k -core decomposition that would allow us to partition the graph and find minimal k -cores within the partitions.

One of the limitations of the algorithm is that it only finds minimal k -cores in undirected graphs and directed graphs are more realistic for real-world applications. However, I can extend the definition of a k -core to directed graphs by considering

the in and out degree of a given graph. Then proceed to find minimal k -cores in the undirected version of the graph utilizing Algorithm 3.4. Finally, check if each of the k -cores obtained from the undirected graph is still a k -core in terms of in or out degree.

The objective of this project was to gain understanding about the k -assembly model and to solve the problem of finding all minimal k -cores of an undirected graph. There is still much to explore in the model of the k -assembly. In particular, it would be interesting to study the k -assembly for a non-fixed value of k . For this approach, it would be necessary to analyze the change in the value of k with respect to time and design a dynamical system on the graph. In terms of the algorithm, a promising research direction is to explore the structure of the graph to minimize the number of times the extension operator is called; this would be extremely helpful for solving the problem on sparse graphs. In general, the problem of finding all minimal k -cores continues to be difficult to solve due to the fact the number of minimal k -cores in a graph grows with the number of vertices and edges. Therefore, any condition to make Algorithm 3.4 backtrack faster or that minimizes the number of times the extension operator is called would be a significant contribution to the solution of the problem.

Chapter 4

The Maximum Weighted Co-2-Plex Problem

4.1 Introduction

A set of pairwise non-adjacent vertices defines a stable set. In the context of applications, the rigid structure of stable sets can be restrictive and sensitive to data error. In response, a recent line of research focuses on stable set relaxations. In this work, we concentrate on a degree-based stable set relaxation called the co-2-plex. The graphs in this paper are all finite, simple, and undirected. Throughout this chapter I utilized first person plural due to the fact that the work presented in this chapter was made in collaboration with Benjamin McClosky.

We present two new algorithms for solving the maximum weighted co-2-plex problem (MWC2P) for $\{\text{claw}, \text{bull}\}$ -free graphs. The MWC2P problem determines a subset of vertices of maximum total weight, in which each node has at most one neighbor in the set. In this article, we show the MWC2P problem for $\{\text{claw}, \text{bull}\}$ -free graphs can be solved in polynomial time. First, we reduce the original graph to one that is claw-free. Then, we utilize Minty's algorithm to solve the maximum weighted stable set problem (MWSSP). In addition, we present a generalization of Minty's algorithm that solves the MWC2P in the original graph. Our results add to the current

line of research focusing on stable set relaxations.

4.2 Basic Terminology and Background

Given a graph $G = (V, E)$, let $G[S]$ denote the subgraph induced by $S \subseteq V$. Define $N_G(v) = \{u \in V \mid uv \in E\}$, $\deg_G(v) = |N_G(v)|$, and $\Delta(G) = \max_{v \in V} \deg_G(v)$ as the neighborhood, degree and maximum degree of a vertex v . Let $\bar{G} = (V, \bar{E})$ be the complement of G , where $e \in \bar{E}$ if and only if $e \notin E$.

A *claw* is a graph consisting of a vertex with three pairwise non-adjacent neighbors. A *paw* is a graph conformed by a triangle and a vertex adjacent to one of the vertices in the triangle. A *bull* is a graph that contains a triangle and two vertex-disjoint pendant edges. A graph is called $\{\text{claw}, \text{bull}\}$ -free if none of its induced subgraphs is a bull or a claw. Figure 4.1 provides a visualization of a claw and a bull graph.

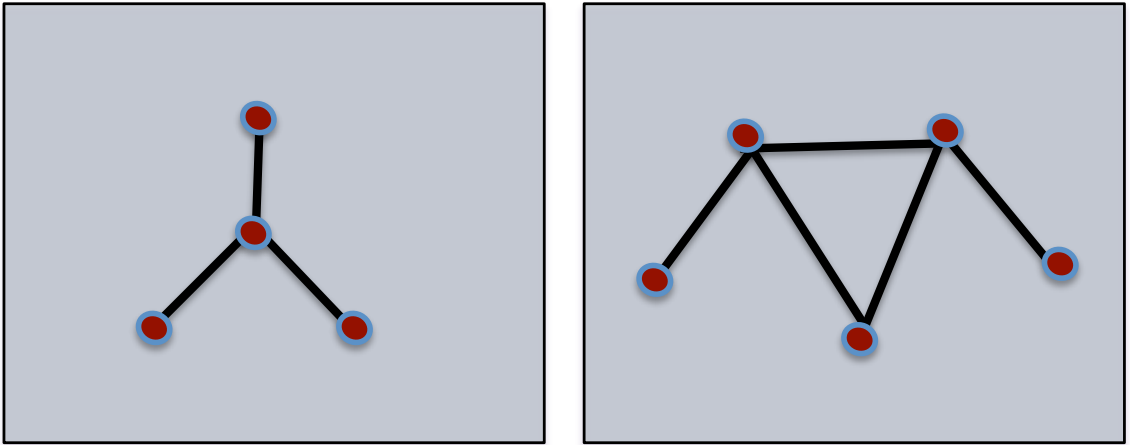


Figure 4.1 : On the left, we see a claw and a bull on the right.

The algorithms presented in this paper assume the given graphs are $\{\text{claw}, \text{bull}\}$ -free.

Definition 4.1 $S \subseteq V$ is a $\text{co-}k\text{-plex}$ if $\Delta(G[S]) \leq k - 1 \ \forall v \in S$ and $k \geq 1$.

A $\text{co-}k\text{-plex}$ in G defines a $k\text{-plex}$ in \bar{G} . In 1978, Seidman and Foster introduced $\text{co-}k\text{-plexes}$ and $k\text{-plexes}$ in the context of social network analysis [47]. A co-1-plex is also known as an independent set or a stable set, a set in which none of the vertices are adjacent. When $k = 2$, we obtain a co-2-plex , which is a subset of vertices in which each vertex is adjacent to at most one other vertex in the set. McClosky and Hicks studied the $\text{co-}k\text{-plex}$ polytope and showed that certain cycles, webs, and claw-like structures induce facets for the co-2-plex polytope. In addition, they used co-2-plexes to characterize a class of integral polyhedra [29]. For a thorough study of the $\text{co-}k\text{-plex}$ polyhedra see [28].

Given a graph G and a set of vertex weights, the maximum weighted co-2-plex (MWC2P) problem identifies the co-2-plex subgraph with the maximum total weight. At the moment, there does not exist a polynomial-time algorithm to solve the MWC2P problem in general. However, there are results that lead us to focus on graphs with a particular structure.

Theorem 4.1 Given G , let \mathcal{K} be the set of maximal 2-plexes in G . The polytope

$$\{x \in \mathbb{Q}^{|V|} : 0 \leq x_v \leq 1, \sum_{v \in K} x_v \leq 2 \ \forall K \in \mathcal{K}\}$$

is integral if and only if G is $\{\text{claw}, \text{paw}\}$ -free and does not contain any induced cycle C^n such that $n \not\equiv 0 \pmod{3}$ [29].

Theorem 4.1 relies on a related structural result which states that any $\{\text{claw}, \text{paw}\}$ -free graph must be a path, cycle, or 2-plex [29]. Due to their simplicity, the $\{\text{claw}, \text{paw}\}$ -free graphs do not warrant further research from an algorithmic perspective. Consequently, we focus on solving the MWC2P problem for $\{\text{claw}, \text{bull}\}$ -free graphs and show the aforementioned graphs define polynomial time instances of the problem. We show that the MWC2P in $\{\text{claw}, \text{bull}\}$ -free graphs reduces to the maximum weighted stable set (MWSS) problem in a claw-free graph. Minty proposed a polynomial time algorithm for solving MWSSP on a claw-free graph [31]. As a result, we present two polynomial time algorithms to solve the MWC2P problem in $\{\text{claw}, \text{bull}\}$ -free graphs.

4.3 A Tractable Instance of the Maximum Weighted Stable Set Problem

Similarly to the MWC2P problem, the MWSS problem determines the stable set subgraph with the maximum total vertex weight. There does not exist a polynomial time algorithm to solve the MWSS problem in general graphs. However, George Minty proposed a polynomial time solution to the MWSS problem for claw-free graphs, since one can determine in linear time whether or not a graph is claw-free [31]. Minty's algorithm was corrected by Nakamura and Tamura in 2001[34]. The algorithm is

based on the observation that whenever a graph is claw-free the symmetric difference between two stable sets will be a path or a cycle. Hence, Minty's algorithm finds augmenting paths to increase the weight of an initial stable set.

Given a weighted claw-free graph, with weight w . Minty's algorithm classifies the vertices of the graph. A vertex is labeled black if it belongs to the stable set and white otherwise. A white vertex is further classified by its number of black neighbors. It is labeled as super-free, free, or bounded if it is adjacent to 0, 1, or 2 black vertices, respectively. Since the graph is claw-free a white vertex can be adjacent to at most two black vertices.

Minty's algorithm proceeds to find maximum weighted augmenting paths between two non-adjacent free vertices. An alternating path is a sequence of vertices in which black and white vertices alternate in adjacency (i.e. W-B-W-B-W). The weight of a path is defined as the weight of white vertices minus the weight of the black vertices. An augmenting path is an alternating path whose weight is positive. A maximum weighted white augmenting path (MWWAP) is one whose weight is maximum between the two free vertices. The symmetric difference between the current stable set and a MWWAP forms a stable set with larger weight. The algorithm continues until it cannot find any MWWAP. Minty proved that a stable set has maximum weight if no maximum augmenting path exists [31]. In addition, he showed that a maximum augmenting path can be found in polynomial time. Hence, Minty's algorithm solves the MWSS problem in polynomial time.

4.3.1 Minty's Algorithm

The algorithms presented on this paper follow the structure Minty's algorithm. The first algorithm reduces the given graph to an instance of the MWSS problem, then it utilizes Minty's algorithm to obtain a solution. The second algorithm utilizes maximum weighted white augmenting paths (MWWAP), that is, paths with white ending vertices and positive weight. It divides its vertex set in two, black and white. Then it finds MWWAPs to increase the weight of a co-2-plex until its weight is maximum among all other co-2-plexes. A more thorough definition of augmenting paths will be given in Section 4.4.2.

Given a weighted claw-free graph, Minty's algorithm classifies the vertices of the graph. A vertex is labeled black if it belongs to the stable set. A vertex is labeled white if it does not belong to the stable set. White vertices are classified by its number of black neighbors. They are labeled superfree, free, or bounded if they are adjacent to 0, 1, or 2 black vertices, respectively. A white vertex cannot be adjacent to three or more black vertices, otherwise it would contradict the fact that the graph is claw-free.

Algorithm 4.1 gives an overview of Minty's algorithm, which is key in understanding the work presented in this paper. Most of the steps described in Algorithm 4.1 are not computationally expensive. Nevertheless, finding the MWWAP between two distinct free vertices is not a simple task. Throughout the rest of this section we will discuss the steps necessary to find a MWWAP.

In order to find a MWWAP between two vertices, Minty proposed a transformation of the original problem into an instance of the maximum weighted matching problem by constructing a very specific graph, called the Edmonds' graph[31] . The celebrated Edmonds blossom algorithm solves the weighted matching problem in polynomial time [17]. Cook and Rohe design an efficient implementation of Edmonds' blossom algorithm [14] [13]. Sbihi generalized the blossom algorithm to find maximum cardinality stable sets in claw-free graphs [45]. The Sbihi and Minty algorithms run in polynomial time and exemplify how the exclusion of an induced subgraph can result in tractable instances of otherwise NP-hard problems.

Algorithm 4.2 provides an outline to find a MWWAP between two distinct free vertices [34]. Nakamura and Tamura confirmed the correctness of algorithm 4.2, but realized that the execution of 2.2.2 contains an error [34] . Before exploring the mistake of Algorithm 4.2, to understand how to find a MWWAP and the construction of the Edmonds' graph, the reader must be introduced to a series of definitions. For more details on the definitions provided in this section see [31] and [34].

Algorithm 4.1

1. $S = \emptyset$
2. **while** S is not optimal
 - 2.1 Classify the white vertices based off
the black vertices
 - 2.2 Find a MWWAP
 - 2.3 **if** no such path exists

break
 - 2.4 $S = S \Delta P$
3. **return** S

A reduced basic structure (RBS) is the reduced graph obtained by ignoring all of the *super free*, *free* vertices except a and b , and all the white vertices that are adjacent to a and b , since they would never appear in an alternating path between a and b . In addition, the RBS includes the weight function w and the semi-optimal stable set S .

A *wing* refers to a nonempty set of bounded vertices, which are adjacent to the same two black vertices x and y . Vertices x_a and x_b are called *regular I*, a black vertex is classified *regular II* if it is adjacent to three or more wings, *irregular* if it is adjacent to exactly two wings, and *useless* otherwise. Let $(v_0, v_1, v_2, \dots, v_{2l})$ be a black alternating path. If $v_2, v_4, \dots, v_{2l-2}$ are all irregular then the subpath $(v_1, v_2, \dots, v_{2l-1})$ is called an irregular white alternating path (*IWAP*) between v_0

and v_{2l} .

Algorithm 4.2

```

2.1 Generate all white alternating paths
    of length 0 and 2

2.2 for each pair of non-adjacent free
    vertices  $a$  and  $b$ 

    2.2.1 Let  $x_a$  and  $x_b$  be the black
        vertices adjacent to  $a$  and  $b$ 

    2.2.2 if  $x_a \neq x_b$ 

        find a MWWAP
        between  $a$  and  $b$ 

    end if

end for

2.3 if all white alternating paths generated
    have nonpositive weight

    return  $S$ 

end if

2.4 Choose MWWAP  $P^*$  among all generated
    white alternating paths

```

The neighbor sets of vertices x_a and x_b are partitioned into two sets, $N^1(x_a) = \{a\}$ and $N^2(x_a) = N(x_a) \setminus \{a\}$, $N^1(x_b)$ and $N^2(x_b)$ are defined in a similar manner. For any *regular II* vertex v , $N(v)$ is uniquely partitioned into $N^1(v)$ and $N^2(v)$ so that

for any x and y in distinct wings adjacent to a vertex v , x is not adjacent to y if and only if one of x and y is in $N^1(v)$ and the other in $N^2(v)$. The aforementioned concepts are necessary to define the Edmonds' graph with vertex set \hat{V} , edge set \hat{E} . In addition, each edge is assigned a color \hat{c} , for more details see Algorithm 4.3.

Algorithm 4.3

Given the RBS

3.1 $\hat{V} = \{\hat{a} = a, \hat{b} = b\}$ and $\hat{E} = \emptyset$

3.2 Label the set of black vertices as:

regular I, regular II, irregular or useless

3.3 Let $x_1 \dots x_r$ be the set of regular I and II vertices

3.4 **for** $i = 1 \dots r$

$$\hat{V} = \hat{V} \cup \{x_i^1, x_i^2\}$$

$$\hat{E} = \hat{E} \cup \{x_i^1 x_i^2\}$$

$$\hat{w}(x_i^1 x_i^2) = w(x_i)$$

$$\hat{c}(x_i^1 x_i^2) = \text{black}$$

end for

3.5 $\hat{E} = \hat{E} \cup \{\hat{a}x_a^1\}$, $\hat{w}(\hat{a}x_a^1) = w(x_i)$

$$\text{and } \hat{c}(\hat{a}x_a^1) = \text{white}$$

3.6 $\hat{E} = \hat{E} \cup \{\hat{b}x_b^1\}$, $\hat{w}(\hat{b}x_b^1) = w(x_i)$

$$\text{and } \hat{c}(\hat{b}x_b^1) = \text{white}$$

3.7 **for** each pair of regular vertices x_i and x_j

for $p, q \in \{1, 2\}$

if there exists an IWAP with ends $\in N^p(x_i)$ and $N^q(x_j)$

$$\hat{E} = \hat{E} \cup \{x_i^p x_j^q\}$$

$$\hat{w}(x_i^p x_j^q) = \max_{x \in IWAP} w(x)$$

$$\hat{c}(x_i^p x_j^q) = \text{white}$$

end if

end for

end for

In the Edmond's graph, a simple path is an alternating path if black and white edges appear alternately, black and white alternating paths are defined similarly to the vertex version. The weight of a path \hat{P} is the sum of the weights of the white edges minus the sum of the weight of the black edges and denote it $\hat{\delta}(\hat{P})$. A maximum weighted matching in the Edmonds' graph gives us a MWWAP in the original graph.

4.3.2 Tamura and Nakamura's Correction

The procedure described in Algorithm 4.3 works as long as the matching consisting of black edges is semi-optimal. Nakamura and Tamura noticed that any graph that does not have a semi-optimal matching contains one of the following subgroups [34] :

1. An augmenting cycle
2. A white-black augmenting path with an unmatched endpoint
3. A black alternating path P and a white alternating path Q such that the endpoints of Q are unmatched, P and Q are vertex disjoint and $\hat{\delta}(P) + \hat{\delta}(Q) > 0$ (referred to as augmenting path pair)
4. A black augmenting path

The Edmond's graph contains only one special case of the aforementioned sub-graphs. That is, augmenting cycles of length less than or equal to 6 [34]. Nakamura and Tamura observed that is only necessary to eliminate augmenting cycles of length 4, while preserving all alternating paths between \hat{a} and \hat{b} to correct the Edmonds'

graph. They pointed out that the Edmonds's graph has no alternating cycle containing (x_a^1, x_a^2) or (x_b^1, x_b^2) , as a consequence one could fix distinct regular II vertices x_i and x_j . Let P_{pq} be the maximum weight *IWAP* corresponding to the edge (x_i^p, x_j^q) in the Edmonds' graph, if such edge exists for $p, q \in \{1, 2\}$, and considered the following cases where an augmenting cycle of length 4 exists:

Case A: there exist both P_{11} and P_{22} , and $\delta(P_{11}) + \delta(P_{22}) > w(x_i) + w(x_j)$

Case B: there exist both P_{12} and P_{21} , and $\delta(P_{12}) + \delta(P_{21}) > w(x_i) + w(x_j)$

A wing W is reachable by irregular vertices to a regular vertex x if there exists an integer $l \geq 1$, distinct irregular vertices z_1, \dots, z_{l-1} and distinct wings $W_1 \dots W_l$ such that W_1 is adjacent to z_1 and W_k is adjacent to z_{k-1} and z_k for $k = 2, \dots, l$ and $z_l = x$. Let $W(x_i, x_j)$ denote the union of all wings that are reachable by irregular vertices by x_i and x_j [34].

The following four lemmas allowed Nakamura and Tamura to correct the Edmonds' graph:

Lemma 4.1 If $N^1(x_j) \subseteq W(x_i, x_j)$, then any white alternating path between a and b in the RBS passes through neither P_{12} nor P_{22} . That is, we can delete the edges (x_i^1, x_j^2) and (x_i^2, x_j^2) from the Edmonds' graph. Similarly if $N^2(x_j) \subseteq W(x_i, x_j)$, we can delete (x_i^1, x_j^1) and (x_i^2, x_j^1) . If $N^1(x_i) \subseteq W(x_i, x_j)$ we can delete (x_i^2, x_j^1) and (x_i^2, x_j^2) . If $N^2(x_i) \subseteq W(x_i, x_j)$ we can delete (x_i^1, x_j^1) and (x_i^1, x_j^2) .

For the following lemma only case A is considered (case B is symmetric). Let us assume none of $N^1(x_i)$, $N^2(x_i)$, $N^1(x_j)$ and $N^2(x_j)$ are contained in $W(x_i, x_j)$.

Lemma 4.2 Paths P_{11} and P_{22} have the same irregular vertices (which may be empty).

Lemma 4.3 If the set of irregular vertices in P_{11} and P_{22} is empty, any white alternating path from a to b does not pass through P_{11} nor P_{22} . Hence, we can delete (x_i^1, x_j^1) and (x_i^2, x_j^2) from the Edmonds' graph (Case B does not occur for the same pair x_i and x_j).

By Lemma 4.2, $P_{11} = (y_1^1, z_1, \dots, y_{l-1}^1, z_{l-1}, y_l^1)$ and $P_{22} = (y_1^2, z_1, \dots, y_{l-1}^2, z_{l-1}, y_l^2)$. Where z_1, \dots, z_l are irregular vertices, y_k^1 and y_k^2 are in the same wing W_k for $k = 1 \dots l$, $y_1^1 \in N^1(x_i)$, $y_l^1 \in N^1(x_j)$ and $y_1^2 \in N^2(x_i)$, $y_l^2 \in N^2(x_j)$.

Lemma 4.4 1. If the set of irregular vertices in P_{11} and P_{22} is the same and not empty. There exists $2 \leq k \leq l-1$ and $y_k^1 = y_k^2$, or there exists $1 \leq k \leq l-1$ such that y_k^1 is not adjacent to y_{k+1}^2 and y_k^2 is not adjacent to y_{k+1}^1 .

2. For such k , let

$$P_{11i} = (y_1^1, z_1, \dots, y_{k-1}^1, z_{k-1}, y_k^1),$$

$$P_{11j} = (y_{k+1}^1, z_{k+1}, \dots, y_{l-1}^1, z_{l-1}, y_l^1),$$

$$P_{22i} = (y_1^2, z_1, \dots, y_{k-1}^2, z_{k-1}, y_k^2),$$

$$\text{and } P_{22j} = (y_{k+1}^2, z_{k+1}, \dots, y_{l-1}^2, z_{l-1}, y_l^2),$$

and let $P'_{12} = (P_{11i}, z_k, P_{22j})$ and $P'_{21} = (P_{22i}, z_k, P_{11j})$. Then $\delta(P'_{12}) + \delta(P'_{21}) = \delta(P_{11}) + \delta(P_{22})$, P'_{12} is an IWAP between $N^1(x_i)$ and $N^2(x_j)$, and P'_{21} is an IWAP

between $N^2(x_i)$ and $N^1(x_j)$.

$$3. \delta(P_{11}) + \delta(P_{22}) = \delta(P_{12}) + \delta(P_{21}).$$

$$4. \delta(P'_{12}) = \delta(P_{12}) \text{ and } \delta(P'_{21}) = \delta(P_{21})$$

The correction of the Edmonds' graph deals with Lemmas 4.1, 4.3 and 4.4. For the first two it is only necessary to delete the unnecessary edges in order to get rid of the unwanted augmenting cycles. For the last case the following steps must be taken into consideration.

1. delete the four edges (x_i^1, x_j^1) , (x_i^2, x_j^2) , (x_i^1, x_j^2) and (x_i^2, x_j^1) ,
2. add two new vertices z_k^i and z_k^j , join them with a black edge and let $\hat{w}((z_k^i, z_k^j)) = \hat{w}(z_k)$.
3. add four white edges (x_i^1, z_k^i) , (x_i^2, z_k^i) , (x_j^1, z_k^j) and (x_j^2, z_k^j) . Assign their weights to be $\hat{w}((x_i^1, z_k^i)) = \delta(P_{11i})$, $\hat{w}((x_i^2, z_k^i)) = \delta(P_{22i})$, $\hat{w}((x_j^1, z_k^j)) = \delta(P_{11j})$ and $\hat{w}((x_j^2, z_k^j)) = \delta(P_{22j})$.

Nakamura and Tamura proved that by applying their revision for every pair of regular II vertices x_i and x_j when case A or case B occurred the edges in the Edmonds' graph form a semi-optimal matching [34]. In addition, they showed that it is still possible to solve MWSS problem for claw-free graphs in polynomial time by adding the correction of the Edmonds' graph. This is due to the fact that the number of cycles of length 4 is polynomially bounded. As a consequence, the revised Edmonds' graph

can be constructed in polynomial time. In the following sections we will introduce two algorithms that are based on this revision of Minty's algorithm to solve the MWC2P problem.

4.4 Algorithms for Finding the Maximum Weighted Co-2-Plex in a {claw, bull}-Free Graph

In this section, we present two algorithms to solve the MWC2P problem in a graph that is {claw, bull}-free. The first one is based on a reduction of the original claw, bull}-free graph into one that is only claw-free. Then we use the revised version of Minty's algorithm to solve the MWSS problem in the reduced graph. This optimal set corresponds to the maximum weighted co-2-plex in the original graph. The reduction as well as the revised version of Minty's algorithm can be done in polynomial time. Hence, the entire algorithm can be executed in polynomial time. The second algorithm follows the structure of the revised version of Minty's algorithm with subtle changes in the vertex labeling process, that allow us to find a co-2plex of maximum weight in the original graph. Let us first introduce the reader to the algorithm based on reduction of the original graph.

4.4.1 The reduction

Given a graph G , this section defines a related graph G' and establishes a correspondence between the set of co-2-plexes in G , \mathcal{I}_G^2 , and the set of stable sets in G' , $\mathcal{I}_{G'}$. De-

fine $\mathcal{V} = V \cup E$ so that the elements of \mathcal{V} consist of singletons and pairs from V . That is, any element $T \in \mathcal{V}$ has the form $T = \{v\}$ for some $v \in V$ or $T = \{u, w\}$ for some $uw \in E$. Let $\mathcal{E} = \{S, T \in \mathcal{V} : S \cap T \neq \emptyset\} \cup \{S, T \in \mathcal{V} : \exists u \in S, \exists v \in T \text{ s.t. } uv \in E\}$, $G' = (\mathcal{V}, \mathcal{E})$, $n = |V|$, $n' = |\mathcal{V}|$, $m = |E|$, and $m' = |\mathcal{E}|$. See Figure 4.2 for an example of a G, G' pair. The fact that $n' = n + m$ implies that the reduction is polynomial. In this section we construct mappings $\Psi : \mathcal{I}_G^2 \mapsto \mathcal{I}_{G'}$ and $\Gamma : \mathcal{I}_{G'} \mapsto \mathcal{I}_G^2$ to show that MWC2P on G reduces to MWSSP on G'

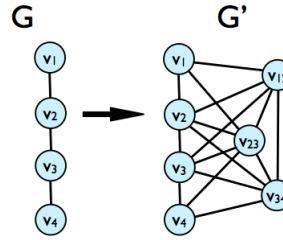


Figure 4.2 : Example of a G, G' pair via the reduction.

Given a co-2-plex $S \subseteq V$, consider the partition $S = S_0 \cup S_1$ where $S_i = \{u \in S : \deg_{G[S]}(u) = i\}$. Note that $u \in S_1$ implies $|N_{G[S]}(u)| = 1$. Let

$$\Psi(u) = \begin{cases} \{u\} & \text{if } u \in S_0, \\ \{u, N_{G[S]}(u)\} & \text{if } u \in S_1. \end{cases} \quad (4.1)$$

Define $\Psi(S) \subseteq \mathcal{V}$ to be the image of S under Ψ . By construction, for all $u \in S$

$$|\{U \in \Psi(S) : u \in U\}| = 1. \quad (4.2)$$

and $U \subseteq S$ for all $U \in \Psi(S)$.

Lemma 4.5 $\Psi(S) \in \mathcal{I}_{G'}$ for all $S \in \mathcal{I}_G^2$.

Proof 4.1. Consider distinct sets $T, W \in \Psi(S)$. Observe that $T, W \subseteq S$ and $T \cap W \neq \emptyset$ together contradict (4.2). Therefore, $T \cap W = \emptyset$.

Now suppose there exists $t \in T$ and $w \in W$ such that $wt \in E$. It follows from $T, W \subseteq S$ that $wt \in E(G[S])$. Consequently, $\Psi(w) = \Psi(t) = \{w, t\}$ and (4.2) together imply $T = W$, a contradiction. Therefore, $TW \notin \mathcal{E}$, and $\Psi(S)$ defines a stable set in G' .

□

Conversely, given a stable set $I \subseteq \mathcal{V}$ in G' , define $\Gamma(I) = \bigcup_{T \in I} T$.

Lemma 4.6 $\Gamma(I) \in \mathcal{I}_G^2$ for all $I \in \mathcal{I}_{G'}$.

Proof 4.2. First note that $\Gamma(I) \subseteq V$. For the assertion to fail, there must exist a vertex of degree at least two in $G[\Gamma(I)]$. Given any triplet $v_1, v_2, v_3 \in \Gamma(I)$, there exist sets $V_1, V_2, V_3 \in I$ such that $v_i \in V_i$. If $V_i \cap V_j = \emptyset$ for each pair $i, j \in \{1, 2, 3\}$, then the definition of \mathcal{E} and $I \in \mathcal{I}_{G'}$ together imply that $\{v_1, v_2, v_3\}$ induces a stable set in $G[\Gamma(I)]$.

Without loss of generality, suppose $V_1 \cap V_2 \neq \emptyset$. Now $I \in \mathcal{I}_{G'}$ implies $V_1 = V_2 = \{v_1, v_2\}$, since otherwise $V_1 V_2 \in \mathcal{E}$. To complete the proof, use $I \in \mathcal{I}_{G'}$ to deduce that $V_3 \cap \{v_1, v_2\} = \emptyset$ and $v_1 v_3, v_2 v_3 \notin E$. It follows that $\Delta(G[\Gamma(I)]) \leq 1$.

□

The following lemma uses the mappings Ψ and Γ to show that MWC2P on G reduces to MWSSP on G' . Given a weight c_v for each $v \in V$, define $c_T = \sum_{v \in T} c_v$ for all $T \in \mathcal{V}$. Using these weights, let z_1^* and z_2^* be the optimal solutions to MWSSP on G' and to MWC2P on G , respectively.

Lemma 4.7 $z_1^* = z_2^*$.

Proof 4.3. Given an optimal weighted co-2-plex $S^* \subseteq V$, Lemma 4.5 implies that $\Psi(S^*)$ is a stable set in G' . Moreover, (4.2) implies that $\sum_{T \in \Psi(S^*)} c_T = \sum_{v \in S^*} c_v = z_2^*$, and so $z_1^* \geq z_2^*$.

Conversely, given an optimal weighted stable set $I^* \subseteq \mathcal{V}$, Lemma 4.6 implies that $\Gamma(I^*)$ is a co-2-plex in G . Furthermore, $I^* \in \mathcal{I}_{G'}$ requires that for every $u \in \Gamma(I^*)$, there exist a unique $T \in I^*$ such that $u \in T$. Therefore, $\sum_{v \in \Gamma(I^*)} c_v = \sum_{T \in I^*} c_T = z_1^*$, and so $z_1^* \leq z_2^*$.

□

Theorem 4.2 MWC2P can be solved in polynomial time on $\{\text{claw}, \text{bull}\}$ -free graphs.

Proof 4.4. Since Lemmas 4.5, 4.6, and 4.7 establish the necessary correspondence between G and G' , it suffices to show that G' is claw-free whenever G is $\{\text{claw}, \text{bull}\}$ -free. From here, simply use Minty's algorithm to solve MWSSP on G' .

Let G be $\{\text{claw}, \text{bull}\}$ -free, but suppose G' contains a claw $\mathcal{C} \subseteq \mathcal{V}$. Let $T \in \mathcal{C}$ be the vertex such that $\deg_{G'[\mathcal{C}]}(T) = 3$. By construction, $T \in \mathcal{V}$ either corresponds to

a singleton or a pair in V . Let $U_1, U_2, U_3 \in \mathcal{C}$ be the vertices satisfying $U_1, U_2, U_3 \in N_{G'}(T)$ and

$$U_1, U_2, U_3 \in \mathcal{I}_{G'}. \quad (4.3)$$

Suppose $T = \{t\}$. There must exist an edge in G' between T and each vertex U_i . By definition of \mathcal{E} , $T \cap U_i \neq \emptyset$ would imply $N_{G'}(T) \subseteq N_{G'}(U_i)$, contradicting (4.3). Therefore, for each i , there exists a $u_i \in U_i$ such that $tu_i \in E$. Observe that $u_1u_2, u_1u_3, u_2u_3 \notin E$ follows from (4.3). Therefore, $\{t, u_1, u_2, u_3\}$ induces a claw in G , a contradiction.

If $T = \{t_1, t_2\}$, then $t_1t_2 \in E$ by construction. Let $i, j, k \in \{1, 2, 3\}$ be distinct indices. Without loss of generality, suppose $t_1 \in U_i$. It follows from (4.3) that $t_1 \notin U_j \cup U_k$ and that $t_1u \notin E$ for all $u \in U_j \cup U_k$. In particular, $t_2 \notin U_j$ and $t_2 \notin U_k$. As before, there exists $u_j \in U_j$ and $u_k \in U_k$ such that $t_2u_j, t_2u_k \in E$, and so $\{t_1, t_2, u_j, u_k\}$ induces a claw in G , a contradiction. Therefore, $t_1 \notin U_i$ implies that $T \cap \{U_1 \cup U_2 \cup U_3\} = \emptyset$, and thus $N_G(t_1) \cup N_G(t_2)$ meets each of U_1, U_2 , and U_3 .

Recall that $\Gamma(\{U_1, U_2, U_3\}) \subseteq V$. Now (4.3) implies that the sets U_i are disjoint and not connected by any edge in E . Therefore, there exists a stable set $\{w, x, y\} \subseteq \Gamma(\{U_1, U_2, U_3\})$ satisfying $\{w, x, y\} \subseteq N_G(t_1) \cup N_G(t_2)$. Clearly $w, x, y \in N_G(t_i)$ would imply that $\{t_i, w, x, y\}$ induces a claw in G . Without loss of generality, notice also that $x, y \notin N_G(t_1)$ implies $x, y \in N_G(t_2)$, which produces the claw $\{t_1, t_2, x, y\}$ in G . Therefore, $|N_G(t_1) \cap \{w, x, y\}| = |N_G(t_2) \cap \{w, x, y\}| = 2$, and $\{t_1, t_2, w, x, y\}$ induces a bull in G , a contradiction.

□

Although the previously presented reduction along with Minty's algorithm solves the MWC2P problem in polynomial time, it is still of interest to present an algorithm that solves the problem in the original graph. Mainly because the graph becomes denser due to the addition of vertices and edges.

4.4.2 Generalization of Minty's Algorithm to Solve the MWC2P Problem in $\{\text{claw}, \text{bull}\}$ -Free Graphs

The algorithm presented in this section solves the MWC2P problem by finding maximum weighted augmenting paths to increase the weight of a co-2-plex until its weight is maximum among all other co-2-plexes. The main difference between this process and the revision of Minty's algorithm is the classification of the vertex set and the fact that in the presented algorithm step 2.2 changes. Instead of only finding a MWWAP between a pair of non-adjacent free vertices, this method also looks for MWWAP between two non-adjacent free type I or super free type II vertices. The definition of free type I and super free type II vertices will be introduced in this section. The algorithm we present includes more items in order to find a co-2-plex. Classifying vertices in the correct manner is essential to the success of the algorithm.

Given a $\{\text{claw}, \text{bull}\}$ -free graph G , a weight function $w : V \rightarrow \mathbb{R}$ and a co-2-plex S . The elements of S conform the black set and the elements of $G \setminus S$ represent the white set of vertices. The algorithm starts with an empty black set S , then it adds

an subtracts elements from S at each iteration. The white vertices are then classified in three categories based on their number of black neighbors. The classifications that we will introduce allow us to find a MWWAP. The symmetric difference between the current set S and a MWWAP produces a co-2-plex with larger weight. The algorithm stops when there are no MWWAPs in the given graph and returns an optimal solution. Now we can proceed to introduce our vertex classification.

Vertex Classification

To determine if a MWWAP exists one must classify the set of white vertices based off their black neighbors in the current set S . Recall, that Minty classified white vertices in the following manner: a white vertex is super free if it is not adjacent to any black vertex, free if it is adjacent to one black vertex and bounded if it is adjacent to two black vertices. In order to generalize Minty's algorithm to solve the MWC2P problem one must introduce a different set of labeling rules for the white vertices.

A vertex that can be added to the current set S without violating the co-2-plex property is called a super free vertex. There are two types of super free vertices. A type I super free vertex does not have any black neighbors. In contrast, a type II super free has exactly one black neighbor. See Figures 4.3 and 4.4 for examples.



Figure 4.3 : A super free vertex of type I



Figure 4.4 : A super free vertex of type 2

A vertex that violates the co2-plex property, but when the symmetric difference between the current set S and that particular vertex gives us a new co-2-plex is called a free vertex. A type I free vertex is one that is adjacent to exactly one black vertex and that black vertex is adjacent to another black vertex. A type II free vertex is adjacent to two black vertices that are not adjacent to any other black vertex. Figures 4.5 and 4.6 show examples of each type of free vertices.



Figure 4.5 : A free vertex of type I



Figure 4.6 : A free vertex of type II

A bounded vertex is adjacent to two, three or four members of S . It cannot be adjacent to more than four black vertices due to the fact that the graph is $\{\text{claw}, \text{bull}\}$ -free. A bounded vertex cannot be added to the current set S without violating the co-2-plex property. Figures 4.7, 4.8, 4.9 display examples of bounded vertices.

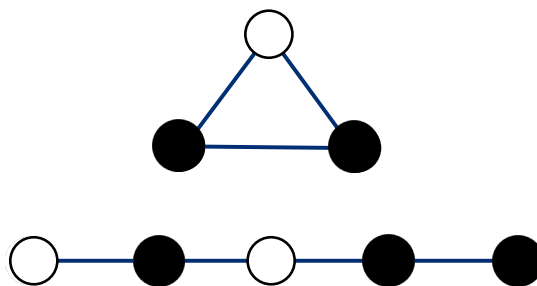


Figure 4.7 : Bounded vertices of type I

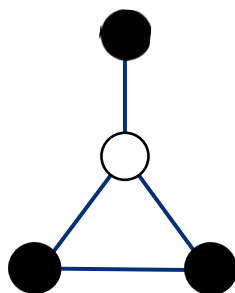


Figure 4.8 : A bounded vertex of type II

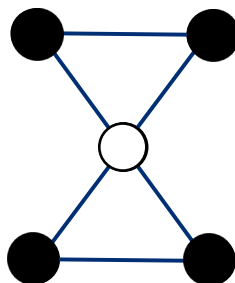


Figure 4.9 : A bounded vertex of type III

4.4.3 Maximum Weighted White Augmenting Paths

Recall that a simple path is a sequence of vertices adjacent to one another without repetition. An alternating path of S is a simple path in which white and black vertices appear alternately. The weight of a path is the sum of the weights of its white vertices minus the sum of the weights of its black vertices and it is denoted by $\delta(P)$. If the weight of an alternating path is positive and its white vertices form a co-2-plex, then it is an augmenting path. If the endpoints of an augmenting path are white, then the path is a white augmenting path. A MWWAP is a white augmenting path having the maximum weight between two distinct free vertices. The super free vertices are considered trivial white augmenting paths. For an illustration of the definitions see Figure 4.10.

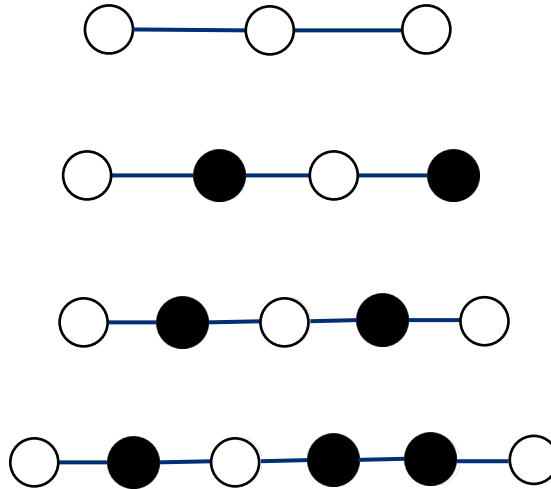


Figure 4.10 : A path, an Alternating Path and two White Alternating Paths

The existence of a MWWAP implies that there is a co-2-plex of larger weight.

Due to the fact that the graph is $\{\text{claw}, \text{bull}\}$ -free, the symmetric difference between the current co-2-plex, S , and the MWWAP, P , forms a co-2-plex with $w(S \Delta P) = w(S) + \delta(P) > w(S)$.

4.4.4 Correctness of the Algorithm

Throughout this section, we will walk the reader to establish the correctness of our proposed algorithm. Recall that the generalization of Minty's algorithm to solve the MWC2P problem follows the structure of Algorithms 4.1 and 4.2, except that step 2.2.2 changes. Instead of finding a MWWAP for each pair of non-adjacent free vertices with different neighborhoods, the proposed method finds MWWAPs between each pair of non-adjacent free vertices of type I and super free vertices of type II with distinct neighborhoods.

Lemma 4.8 *Given a $\{\text{claw}, \text{bull}\}$ -free graph G , let P be a MWWAP generated by the generalization of Minty's algorithm. Then the white vertices in P are not adjacent to any black vertices outside P .*

Proof 4.5. *By definition of MWWAP, we know that a white vertex is adjacent to at most one white vertex in P . Let $u \in P$ be a white vertex and S be the co-2-plex of black vertices.*

If u is adjacent to another white vertex in P , the following cases apply:

If u is a super free of type II vertex, by definition it is adjacent to exactly one black

vertex v , since P is an alternating path $v \in P$, which implies u is not adjacent to black vertices outside P .

If u is an end of P and a free type I vertex, it is adjacent to one of two consecutive black vertices. Since the proposed algorithm is restricted to find MWWAPs of length greater than two between non-adjacent vertices, $\tilde{u} \in N(u)$ and P is not an end. If the consecutive black vertices are in P . We get that u is adjacent to a black and a white non consecutive vertices in P , which implies that u cannot be adjacent to a black vertex in the co-2-plex S unless there is a claw in G .

If u is an end of P , a free type I vertex and only the black vertex that is adjacent to u is in P . Since \tilde{u} is not an end, if \tilde{u} and u share the same black neighbor, there exists a black vertex in S that is only adjacent to \tilde{u} . In summary, u , \tilde{u} and a black vertex form a triangle with two vertex-disjoint pendant edges, which gives a bull. If \tilde{u} and u do not share a black neighbor, then the black neighbor of u has three pairwise non-adjacent neighbors, one black and two white, which gives a claw.

If u is not an end of P , u has two black neighbors. If both of those neighbors are not adjacent to any black vertex outside P and u has a black neighbor outside of P , since S is a co-2-plex, we get that G has a claw. If one of the black neighbors shares a black neighbor with u outside P , the u has two neighbors in P and one not in P that are pairwise non-adjacent, which gives a claw.

If u is not adjacent to any other white vertex in P , the following cases apply:

If u is an end of P , then it could not be adjacent to a black vertex outside P , otherwise it would not be a free type I or super free type II vertex.

If u is not an end of P , then it is adjacent to two black vertices in P . If both of those neighbors are not adjacent to any black vertex outside P and u has a black neighbor outside of P , then we get a claw. If one of the black neighbors shares a black neighbor with u outside P , there is a white vertex in P that is adjacent to that particular black neighbor, which gives a bull.

By now, we have exhausted all the possible cases. We get that a white vertex in a MWWAP P , generated by the extension of Minty's algorithm is not adjacent to any black vertices outside P unless G has a claw or a bull.

□

Theorem 4.3 A MWWAP ensures the existence of a co-2-plex with larger weight.

Proof 4.6. Let $S^{(i)}$ be a co-2-plex at an arbitrary iteration i and let P be a MWWAP. Since the graph is $\{\text{claw}, \text{bull}\}$ -free, by Lemma 4.8 the white vertices in P are not adjacent to any black vertex outside P . Then $S^{(i+1)}$ is equal to the symmetric difference $S^{(i)} \Delta P$, which is composed of two co-2-plexes. One is the set of white vertices in P and the other is the set of black vertices in S and not in P . Since the two sets do not

have edges in common, the degree of all the vertices in $S^{(i+1)}$ is less than or equal to one. In addition, the fact that P is a MWWAP implies $\delta(P) > 0$. As a consequence, $w(S^{(i+1)}) = w(S^{(i)}) + \delta(P) > w(S^{(i)})$.

□

4.5 Numerical Results

The presented algorithms were implemented using C++ and tested in a MacBook Pro with an Intel Core i5 2.4 GHz Processor. The matching problems were solved with the aid of Blossom4 [13]. We observed that the complement of {claw, bull}-free graphs contain triangles as induced subgraphs, see Figure 4.11. Hence, we utilized the complement of triangle-free graphs to test the performance of our algorithms.

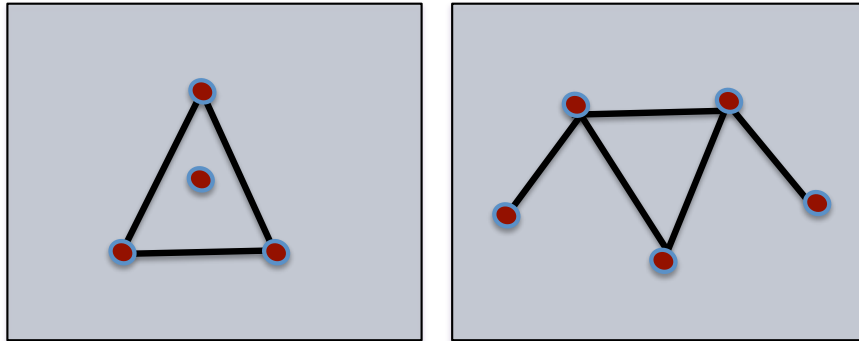


Figure 4.11 : The Complement of a Claw and a Bull

Among well known families of triangle-free graphs are the Mycielski graphs. Table 4.1 displays the performance of our algorithms when tested using the complement of

Mycielski graphs.

n	m	Algorithm 4.4.2	Reduction Algorithm
5	5	0.001 s	0.001 s
11	35	≈ 0	0.001 s
23	182	0.001 s	0.005 s
47	845	0.003 s	0.037 s
95	3710	0.009 s	0.621 s
191	15785	0.027 s	13.46 s
383	65882	0.086 s	567.85 s
767	271565	0. 05 s	—
1535	1109990	1.149 s	—
3071	4510385	4.65 s	—
6143	18251282	18.04 s	—
12287	73631285	296.34 s	—

Table 4.1 : Performance of Algorithm 4.4.2 and the Reduction Algorithm on the complement of Mycielski graphs

Another two less popular triangle-free graphs are the complement of a butterfly and a triangle graph, see Figure 4.12. Due to the fact that both of them are the complement of triangle-free graphs and are perfect, the Replication Lemma allows us to generate more test instances for our problem utilizing these two graphs as our base.

Given a graph G , we created a new graph G^* by adding a new vertex v^* adjacent to v and all of its neighbors for all $v \in V$, for more details about generating graphs with the replication lemma see [26]. Tables 4.2 and 4.3 display the results of testing our algorithms with graph generated by the Triangle and the Butterfly graph.

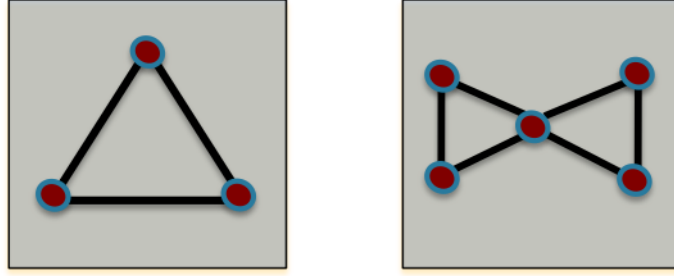


Figure 4.12 : A Butterfly and a Triangle are triangle-free graphs in the complement

In addition, we used the House of Graphs Data Base to obtain their collection of acyclic graphs, which are triangle-free. Since, all acyclic graphs are bipartite and a bipartite graph is perfect. By the Perfect Graph Theorem, the complement of a perfect graph is perfect [26]. As a consequence, we used the complement of acyclic graphs as test cases. The collection of acyclic graphs only contained graphs of cardinality less than 50 and the MWC2P problem was solved within fractions of a second. However, in the near future we will use the collection of acyclic graphs to generate larger test instances with the aid of the Replication Lemma.

Overall, Algorithm 4.4.2 outperforms the reduction method. Mainly because the reduction method transforms the given graph to a larger one, then it proceeds to

n	m	Algorithm 4.4.2	Reduction Algorithm
6	12	0.002179 s	0.001816 s
12	42	0.000723 s	0.000747 s
24	138	0.004731 s	0.013163 s
48	438	0.003772 s	18.5314 s
96	1362	0.001 s	90.127973 s
192	4182	0.006719 s	—
384	12738	187.117 s s	—

Table 4.2 : Performance of Algorithm 4.4.2 and the Reduction Algorithm on graphs generated with a triangle as base for replication

n	m	Algorithm 4.4.2	Reduction Algorithm
10	23	0.002975 s	0.003929 s
20	79	0.00348 s	0.876278 s
40	257	0.02995 s	3172.91 s
80	811	0.684186 s	—
160	2513	32.8705 s	—
320	7699	775.008 s	—

Table 4.3 : Performance of Algorithm 4.4.2 and the Reduction Algorithm on graphs generated with a butterfly as base for replication

apply Minty’s algorithm on the new graph. The increment in the number of nodes and edges is directly proportional to the number of free vertices considered to find MWWAPs in Algorithm 4.2. As a consequence, the number of times a matching problem must be solved increases.

4.6 Discussion

In this chapter, we introduced two polynomial time algorithms for solving the MWC2P problem for {claw, bull}-free graphs. Both algorithms are based off Minty’s algorithm for solving MWSSP and contribute to the study of stable set relaxations. Algorithm 4.4.2 operates directly on the given graph G , while the reduction method takes an indirect approach.

An interesting path of research would be to generalize the proposed algorithms to larger values of k . This would likely increase the number of forbidden graphs. For instance, Minty’s algorithm only requires the graph to be claw-free to solve MWCKP when $k = 1$ in polynomial time. The proposed algorithms solve the problem for $k = 2$ and have two forbidden graphs.

Chapter 5

The Minimum k -core Problem

In the area of combinatorial optimization, one of its most famous classical problems is the maximum clique problem. It asks for the clique of maximum cardinality. Extensive work in understanding its polyhedral structure and complexity has been made. The maximum clique problem was among the first problems shown to belong to the class of NP-complete problems [22]. In addition, it has multiple integer programming (IP) formulations. For a comprehensive literature review on this problem see [4].

The maximum k -core problem is a natural generalization of the maximum clique problem. It exemplifies how it may be convenient to use clique generalization models. As opposed to the clique problem, the maximum k -core problem is solvable in polynomial time via Algorithm 3.1 [2].

The minimum k -core problem asks for a k -core of minimum cardinality. Interest on this problem emerge from the relation between the minimal k -core problem and the theory of cell assemblies presented in Chapter 3. The fact that a minimum k -core is minimal prompt researchers to model it as an IP [57]. Nevertheless, it is important to emphasize that a minimal k -core is not necessarily minimum.

In contrast to the maximum clique problem, the amount of known results regarding the polyhedral properties of the minimum k -core problem is scarce. Throughout this

chapter the reader will be walk through a study of the k -core polytope.

5.1 Formulation of the Binary Integer Program

One way to solve the minimum k -core problem is to model it as a binary integer program. That is, an optimization problem with a linear objective function, linear constraints and variables that take integer values of zero or one. Throughout this chapter the reader will be guided through my approach to solve the aforementioned IP.

Let $G = (V, E)$ be an undirected graph, A_k be the adjacency matrix of the given graph G and $x \in \{0, 1\}$ the incidence vector of minimum k -cores. Note that for a given positive integer k ,

$$A_k x \geq kx \quad \Rightarrow \quad 0 \geq (kI - A_k)x$$

the zero vector can be a solution unless the following inequality is added:

$$\sum_{i=1}^n x_i \geq 1$$

adding this inequality gives us the following binary integer program:

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ \text{s.t.} \quad & Ax \leq b \\ & x_i \in \{0, 1\} \end{aligned} \tag{5.1}$$

$$A = \begin{pmatrix} kI - A_k \\ -1 \dots -1 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}.$$

In formulation 5.1, for a graph with $|V| = n$, the first n constraints ensure the solution is a k -core. The $n+1$ constraint forbids the zero vector from being a solution.

To solve Problem 5.1, I used the fact that a linear program (LP) is a relaxation of an IP and relaxed the binary variables. Nevertheless, the IP relaxation may not yield to an integral solution. Through the rest of this chapter I introduce a combination of ideas and techniques utilized to obtain an integral solution.

5.2 Linear Relaxation and Convex Hull

A linear relaxation of an IP formulation is the LP obtained by removing the integer constraints.

$$\begin{array}{ll} \min & \sum_{i=1}^n x_i \\ \text{s.t.} & Ax \leq b \\ & x_i \in \{0, 1\} \end{array} \qquad \begin{array}{ll} \min & \sum_{i=1}^n x_i \\ \text{s.t.} & Ax \leq b \\ & x_i \in [0, 1] \end{array} \quad (5.2)$$

Formulation 5.2 displays the original IP formulation along with its relaxation on the right hand side. In the relaxation, I allow the the variables to take fractional values in the interval $[0, 1]$. Although the solution of the relaxation in 5.2 may not be

a solution to Problem 5.1, it is still of interest since it is a lower bound of the original problem. Moreover, if the optimal solution of the LP relaxation is integral, then it is the solution of Problem 5.1.

Example 5.1 One instance in which the relaxed formulation in 5.2 does not yield to an integral solution is if we consider the complete graph on four nodes K_4 and $k = 2$. The relaxation yields to the fractional solution $x = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)^t$, which is not a solution of Problem 5.1. Although the fractional solution is far from the optimal integer solution $x = (1, 1, 0, 1)^t$, it does tell that the objective function of the problem must be at least one and is a starting point to solve Problem 5.1 for K_4 .

To solve the IP formulation in Problem 5.1 the reader must be walked through a series of definitions. For more details on the definitions introduced on this chapter see [54].

Definition 5.1 A subset of \mathbb{R}^n described by a finite set of linear constraints $P = \{x \in \mathbb{R}^n | Ax \leq b\}$ is a polyhedron.

A polytope is a bounded polyhedron. As in linear programming, the constraint set of the IP forms a polytope, but the feasible region is not a convex set. This is due to the fact that the feasible region of an IP is given by the integer points in the polytope instead of the entire polytope.

Throughout this chapter I will refer to the polytope associated with Problem 5.1 as the k -core polytope. The reader would see that different formulations of Problem

5.1 yield different linear relaxations. Some of these relaxations are better than others, and ideally I would like to find a formulation in which each extreme point is integral. Nonetheless, the fact that there is an infinite number of formulations to describe the same problem makes the task of finding the ideal formulation difficult.

Definition 5.2 Given a set $X \subseteq \mathbb{R}^n$, the convex hull of X is

$$\text{conv}(X) = \{x | x = \sum_{i=1}^t \lambda_i x^i, \lambda \in \Lambda\}$$

over all finite subset $\{x^1, \dots, x^t\}$ of X

$$\Lambda = \{\lambda \in \mathbb{R}^t | \sum_{i=1}^t \lambda_i = 1, \lambda_i \geq 0\}$$

The convex hull of X is a polyhedron and all of its extreme points lie in X . One can think about it as the smallest polyhedron containing X . In Problem 5.1, $X = \{x \in \{0, 1\}^n | Ax \leq b\}$. As a consequence, Problem 5.1 is equivalent to the following LP.

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ \text{s.t.} \quad & x \in \text{conv}(X) \end{aligned} \tag{5.3}$$

In theory, solving Problem 5.3 gives a solution to Problem 5.1. In practice, there may be an exponential number of linear inequalities necessary to represent $\text{conv}(X)$. Hence, in most cases is not possible to use Problem 5.3 directly to find a solution. However, an approach to approximate $\text{conv}(X)$ in a neighborhood of an optimal solution will be discussed in this chapter.

Since solving Problem 5.3 directly is not a practical option, I consider the separation problem. That is, given x^* in \mathbb{R}^n , is x^* in $\text{conv}(X)$? If x^* is not in $\text{conv}(X)$, then find an inequality $\pi x \leq \pi_0$ satisfied by all points in X , but violated by x^* . This problem is of particular importance to find a solution of Problem 5.1. However, I must discuss a techniques to solve an integer program first.

5.3 Branch and Bound

Branch and bound (B&B) is the most popular method used to solve an IP. It is a divide and conquer approach coupled with a pruning of the search space using the LP relaxation. This section walks the reader through an example that demonstrates the approach. The following IP is Problem 5.1 for K_4 and $k = 2$:

$$\begin{aligned}
 z &= \min x_1 + x_2 + x_3 + x_4 \\
 s.t. \quad & 2x_1 - x_2 - x_3 - x_4 \leq 0 \\
 & -x_1 + 2x_2 - x_3 - x_4 \leq 0 \\
 & -x_1 - x_2 + 2x_3 - x_4 \leq 0 \\
 & -x_1 - x_2 - x_3 - 2x_4 \leq 0 \\
 & -x_1 - x_2 - x_3 - x_4 \leq -1 \\
 & x_i \in \{0, 1\} \quad i = 1 \dots 4
 \end{aligned} \tag{5.4}$$

Let S be the feasible region of problem 5.4. To obtain the first lower bound one must relax the integrality constraints such that $x_i \in [0, 1]$, solve the LP relaxation and set the upper bound $\bar{z} = \infty$. The solution gives the lower bound $\underline{z} = 1$ and the nonintegral solution $\underline{x} = (\frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3})^t$. Clearly, \underline{x} is not in S .

Due to the fact that $\underline{z} < \bar{z}$ one must split up the feasible region utilizing a technique called branching. That is, choose an integer variable that is the most fractional out of the basic variables in the LP relaxation and split the problem in two. In Problem 5.4, $\underline{x}_1, \underline{x}_2$ and \underline{x}_3 are equally fractional. \underline{x}_1 is selected arbitrarily to obtain the following subproblems:

$$S_1 = S \cap \{x \mid x_1 \leq \lceil \underline{x}_1 \rceil\}$$

$$S_2 = S \cap \{x \mid x_1 \leq \lfloor \underline{x}_1 \rfloor\}$$

Note that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$. Because $x_1 \geq 0$ and $x_1 \leq 1$ and the subproblems are mutually exclusive, $S_1 = S \cap \{x \mid x_1 = 1\}$ and $S_2 = S \cap \{x \mid x_1 = 0\}$.

Subproblems S_1 and S_2 are associated with a node in the search tree, see Figure 5.1.

Their respective nodes are active since they have not been explored.

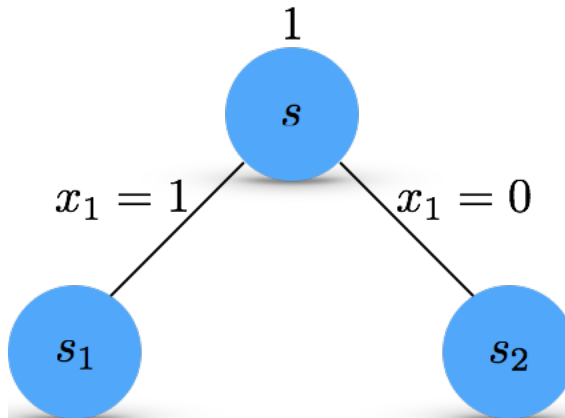


Figure 5.1 : Partial branch and bound tree for Problem 5.4

The following step is to choose a node to be examined for the list of active problems.

I choose S_1 arbitrarily and reoptimize via dual simplex algorithm to obtain $\underline{z}_1 = 3$

as new lower bound and $\underline{x}^1 = (1, 1, 0, 1)^t$ as a solution. S_1 lead us to an integral solution, the best feasible solution is updated $\bar{z} \leftarrow \min\{\bar{z}, 3\}$ and \underline{x}^1 is stored. S_1 is pruned by optimality as shown in Figure 5.2.

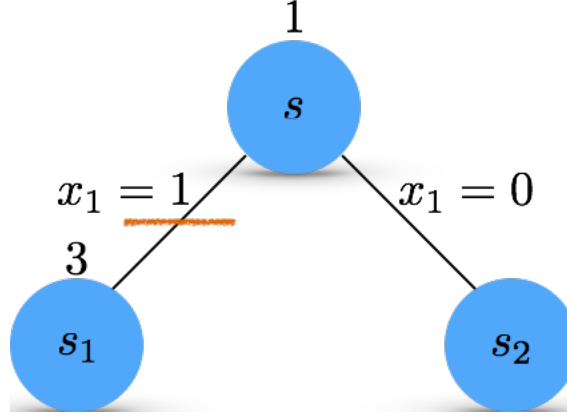


Figure 5.2 : The node associated with S_1 is pruned by optimality

Now the node list only contains S_2 , reoptimizing the problem gives us $\underline{x}^2 = (0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})^t$ and $\underline{z}_2 = 1$. Let us branch on \underline{x}_3^2 and consider its associated subproblems $S_{21} = S \cap \{x \mid x_1 = 0, x_3 = 1\}$ and $S_{22} = S \cap \{x \mid x_1 = 0, x_3 = 0\}$. Figure 5.3 shows the associated search tree with this step of the branching process.

Arbitrarily select S_{21} , the resulting linear program has optimal solution $\underline{x}^{21} = (0, 1, 1, 1)^t$. The obtained solution is integral, but $\underline{z}_{21} = 3$ which implies $\bar{z} = \underline{z}_{21}$, the node is pruned by bound as shown in Figure 5.4.

select S_{22} is not feasible and is pruned by infeasibility. Now, the node list is empty and the algorithm terminates. The optimal solution is $\underline{x} = (1, 1, 0, 1)^t$ with objective function $z = 3$. In Figure 5.5, the reader can see the full branch and bound search

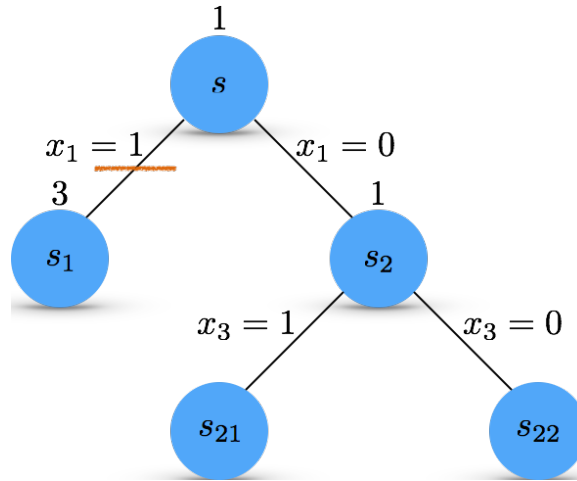


Figure 5.3 : Partial branch and bound tree for Problem 5.4

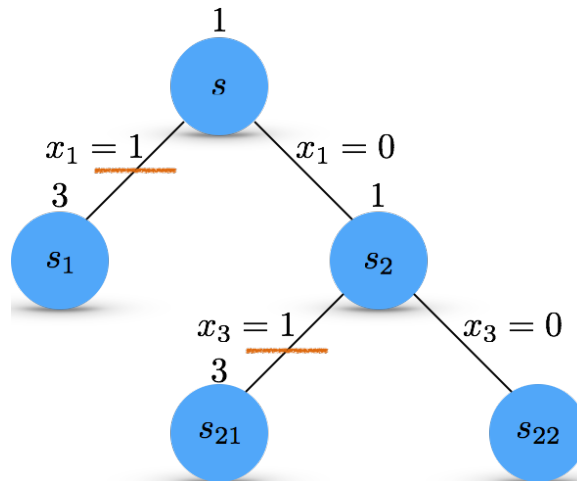


Figure 5.4 : The node associated with S_{21} is pruned by bound

tree of Problem 5.4.

Branch and Bound worked well for solving Problem 5.4. However, the solution of difficult IPs requires a combination of branch and bound and other ideas that would be

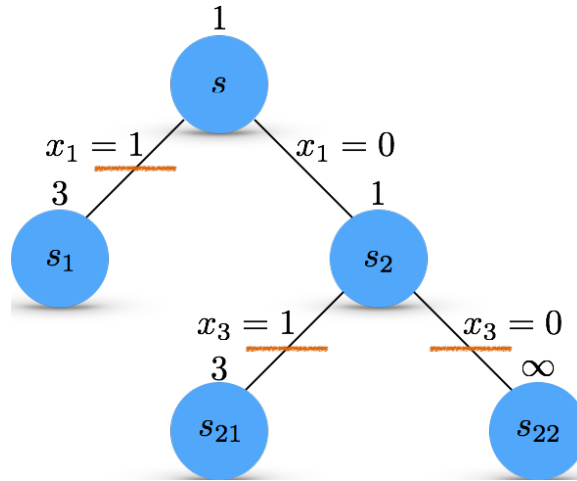


Figure 5.5 : Complete branch and bound tree for Problem 5.4

introduced in this chapter. For instance, one may consider utilizing branch and bound along with an approximation of the convex hull in the neighborhood of an optimal solution. In order to do this, the reader must be introduced to the fundamental concept of a valid inequality.

5.4 Valid Inequalities and Cutting Plane Algorithms

Let us recall that Problem 5.1 is equivalent to the LP relaxation presented in Problem 5.3. That is, to solve the IP is equivalent to solving the LP relaxation in the convex hull. Unfortunately, due to the nature of Problem 5.1 it is very difficult to find a good description of $\text{conv}(X)$. However, given an instance of Problem 5.1 one can try to approximate $\text{conv}(X)$ for the given instance. In order to achieve an approximation, one must be introduced to the concept of a valid inequality. That is, an inequality

that is satisfied by the whole set X .

Definition 5.3 An inequality $\pi x \leq \pi_0$ is a valid inequality for $X \subseteq \mathbb{R}^n$ if $\pi x \leq \pi_0$ for all $x \in X$.

The inequalities $a_i x \leq b_i$ for $i = 1 \dots n + 1$ are trivial valid inequalities of X . If $X = \{x \in \{0, 1\}^n | Ax \leq b\}$ and $\text{conv}(X) = \{x \in \mathbb{R}^n | \bar{A}x \leq \bar{b}\}$ the constraints $a^i x \leq b$ and $\bar{a}^i x \leq \bar{b}$ are valid inequalities for X . To determine when an inequality is valid for X one must use the fact that if $\bar{X} = \{y \in \mathbb{Z}^1 | y \leq b\}$, then the inequality $y \leq \lfloor b \rfloor$ is valid for \bar{X} .

The separation problem is the following: Given x^* in \mathbb{R}^n , is x^* in $\text{conv}(X)$? If x^* is not in $\text{conv}(X)$, then find a valid inequality for X that is violated by x^* .

Definition 5.4 A cut is a valid inequality that separates the current fractional solution x^* .

The cutting plane algorithm in its general form can be used to generate cuts. It starts by solving the LP relaxation. If the solution is integral it stops. Otherwise, it finds a valid inequality that will exclude x^* and solves the LP relaxation again. A more detailed explanation is given in Algorithm 5.1 [54].

If Algorithm 5.1 terminates and does not find an integral solution for the IP, $X_R^t = X_R \cap \{x | \pi^i x^i \leq \pi_0^i \ i = 1 \dots, t\}$ is an improved formulation that can be given as an input to a branch and bound algorithm. Although most of the steps in Algorithm 5.1 seem straight forward, the reader must still be introduced to a general procedure

to find a valid inequality.

Algorithm 5.1 [Cutting Planes]

1. Initialization. Set $t = 0$ and $X_R^0 = X_R$

2. Iteration t . Solve the LP:

$$z = \min\{c^T x \mid x \in X_R^t\}$$

Let x^t be an optimal solution.

If $x^t \in \mathbb{Z}^n$, stop. x^t is an optimal solution.

Else solve the separation problem for x^t .

If an inequality to cut x^t is found such that $\pi^t x^t > \pi_0^t$,

$$\text{set } X_R^{t+1} = X_R^t \cap \{x \mid \pi^t x^t \leq \pi_0^t\}.$$

Else stop.

The Chvátal-Gomory (C-G) procedure was the first one to be shown to generate all valid inequalities for an IP in a finite number of iterations. The valid inequalities obtained from this procedure are known as C-G cuts.

The C-G procedure to construct a valid inequality for the set $X = X_R \cap \mathbb{Z}^n$, where A is an $m \times n$ matrix with columns $\{a_1, a_2, \dots, a_n\}$, and $u \in \mathbb{R}_+^m$ is the following:

1. the inequality

$$\sum_{j=1}^n u a_j x_j \leq ub$$

is valid for X_R as $u \geq 0$ and $\sum_{j=1}^n a_j x_j \leq b$,

2. the inequality

$$\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq ub$$

is valid for X_R as $x \geq 0$,

3. the inequality

$$\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$$

is valid for X as x is integral, and $\sum_{j=1}^n \lfloor ua_j \rfloor x_j$ is integral.

This simple procedure can be used to generate every valid inequality of X . A more specific approach to cutting planes is one that utilizes C-G cuts and is known as Gomory's Fractional Cutting Plane Algorithm.

Given X_R , it is possible to rewrite the problem in standard form by adding slack variables. In the following formulation, I represents the identity matrix and x_S the vector of slack variables.

$$\min\{c^T x \mid Ax + Ix_S = b, x \geq 0, x \in \{0, 1\}, x_S \geq 0\}$$

First, one must solve the associated LP relaxation and find an optimal basis B . The matrix A and vector x are then partitioned into two submatrices B and N and vectors x_B and x_N , associated with the columns of basic and nonbasic variables respectively. The fact that B is a nonsingular matrix allows to derive a C-G cut.

$$Ax = [B \quad N]x$$

$$Bx_B + Nx_N = b$$

Since B is invertible,

$$x_B + B^{-1}Nx_N = B^{-1}b$$

The fact that $x \geq 0$ and integer implies that,

$$x_B + \lfloor B^{-1}N \rfloor x_N \leq B^{-1}b$$

$$x_B + \lfloor B^{-1}N \rfloor x_N \leq \lfloor B^{-1}b \rfloor$$

For each fractional variable it is possible to find a C-G cut. One must choose a fractional basic variable to do it. Given an optimal basis B , recall that $x_N = 0$ which implies that $x_B = B^{-1}b$. Let j be the index of a fractional basic variable, and i the index of the constraint corresponding to variable j . Then the cut for variable j is the following:

$$x_j + \sum_{l \in N} \lfloor (B^{-1}N)_{il} \rfloor x_l \leq \lfloor (B^{-1}b)_i \rfloor$$

In theory, one can use the Gomory's Fractional Cutting Plane Algorithm as general procedure to generate all non-trivial valid inequalities to generate cutting planes and solve the LP relaxation. However, this would be equivalent to approximating $\text{conv}(X)$, and it may involve finding cuts that slowly lead to an optimal integer solution.

In this work, I studied the structure of the C-G cuts generated by the general procedure in Problem 5.1. Through numerical experiments, I observed that vertices on a minimal cycle transversal and a minimal vertex cover of G define valid inequalities

of X . The following definitions are necessary to formally introduce the aforementioned valid inequalities for X .

Definition 5.5 A cycle on three or more vertices is a simple graph whose vertices can be arranged in a cyclic sequence in such a way that two vertices are adjacent if they are consecutive in the sequence and are nonadjacent otherwise.

A cycle on one vertex consists of a single vertex with a loop, and a cycle on two vertices is composed by two vertices joined by a pair of parallel edges. These are valid definitions of a cycle; yet, they are not considered in this section since I only deal with simple graphs in my formulation. Hence, the reader must only consider Definition 5.5 when discussing cycles.

Definition 5.6 A cycle transversal of a graph G is a set of vertices which meets every cycle in G .

One example of a cycle transversal on K_4 is its vertex set V and every subset of V . Since the graph has the maximum number of possible edges, any singleton with one node represents a cycle transversal. A cycle transversal is called minimal if none of its proper subsets is itself a cycle transversal. Although the whole vertex set and any of its subsets on K_4 is a cycle transversal, only the subsets with cardinality one are minimal.

Definition 5.7 An edge transversal or vertex cover is a subset \tilde{E} of V such that every edge G has at least one end in \tilde{E} .

As in the cycle transversal example on K_4 , the whole vertex set V and any of its subsets are vertex covers. A cover is called minimal if none of its proper subsets is itself a cover, and only subsets with cardinality one are minimal vertex covers of K_4 .

In Problem 5.1, I aim to find a minimum k -core. To explore the polyhedral structure of the problem I relax the integer requirements for $x \in X$ and define $X_R = \{x \in \mathbb{R}^n | Ax \leq b\}$. To strengthen the relaxation X_R , I introduce some valid inequalities. Note that for $k \geq 2$, every vertex in the k -core has degree at least 2. This observation lead me to conclude that a minimal cycle transversal is a valid inequality for the convex hull of k -cores. Similarly, every vertex in a k -core for $k \geq 1$ has degree at least one. As a consequence, a minimal vertex cover is a valid inequality for the convex hull of k -cores.

Theorem 5.1 \forall minimal cycle transversal C and $k \geq 2$, the inequality $\sum_{u \in C} x_u \geq 1$ is valid for X .

Proof 5.1 Let G be an arbitrary simple graph and x be an arbitrary element of X . Then the entries of x contain the vertices that are part of one minimum k -core K of G . By definition of k -core for $k \geq 2$, every vertex in a k -core has degree at least 2. This implies that the entries of x represent a set of vertices with minimum degree greater than or equal to 2.

Let $P := \tilde{x}_0 \tilde{x}_1 \dots \tilde{x}_{l-1} \tilde{x}_l$ be a longest path in $G[K]$. Since $\deg(\tilde{x}_l) \geq 2$, it must have another neighbor besides \tilde{x}_{l-1} , call it \bar{x} . If \bar{x} is not in P , then $\bar{P} := \tilde{x}_0 \tilde{x}_1 \dots \tilde{x}_{l-1} \tilde{x}_l \bar{x}$

is a path of $G[K]$ and contradicts the fact that P is a longest path. Hence, $\bar{x} = \tilde{x}_i$ for some $i \in [0, l - 2]$ and \bar{P} is a cycle.

Therefore, the subgraph induced by K contains a 2-core which is a cycle. Since $X \neq \emptyset$ we know that the cardinality of our set C is at least one. Then $\sum_{u \in C} x_u \geq 1$ for all $x \in X$.

□

Theorem 5.2 \forall minimal vertex cover \tilde{C} the inequality $\sum_{u \in \tilde{C}} x_u \geq 1$ is valid for X .

Proof 5.2 Let G be an arbitrary simple graph and x be an arbitrary element of X . Then the entries of x contain the vertices that are part of one minimum k -core K of G . By definition of k -core, every vertex in a k -core has degree at least 1. This implies that $G[K]$ must contain an edge. Therefore, the subgraph induced by K contains a 1-core which is an edge.

Again, since $X \neq \emptyset$ we know that the cardinality of our set \tilde{C} is at least one. Then $\sum_{u \in \tilde{C}} x_u \geq 1$ for all $x \in X$.

□

A cycle transversal can be found utilizing Depth First Search (DFS). Given a graph G and a chosen root node, DFS traverses all vertices of the graph. One must consider the back edges of the spanning tree obtained by applying DFS. That is edges in the

tree that join a node in the tree to one of its ancestors, but are not part of the spanning tree. Then the set of ancestors touched by the set of back edges forms a cycle transversal. If one is only interested in a cycle transversal of all the cycles containing a vertex in particular, then one can choose such a vertex as the root of the DFS tree. In that case, one must only consider the back edges that join vertices in the tree to that particular vertex. The descendants of the chosen vertex that are joined to it by a back edge form a cycle transversal. Although this approach does not guarantee it would find cycle transversal with a minimal property, I use the fact that one must be contained within the set.

One can find a vertex cover VC greedily. Set $\tilde{G} := G$, $VC = \emptyset$ and $VE = V$. Then choose the vertex of maximum degree v and set $VC = VC \cup \{v\}$, $VE = V \setminus \{v\}$ and $\tilde{G} := \tilde{G} \setminus \{v\}$. Proceed in this manner until $VE = \emptyset$, then VC is vertex cover.

A popular question when discussing valid inequalities is: how can one select a useful valid inequality? The two inequalities introduced in this section are valid. Nevertheless, through numerical experiments I observe that adding vertex cover inequalities by finding a vertex cover as it was described in this chapter does not help significantly to solve Problem 5.1. In contrast, adding cycle transversal inequalities did help to get a tighter formulation. Now the reader must be introduced to Branch and cut in order to understand the addition of valid inequalities to branch and bound.

5.5 Branch and Cut

A branch and cut algorithm follows the structure of branch and bound. However, it refines the LP relaxation by adding cutting planes each time it branches. Although adding cuts at each node increases the amount of work per node, the addition of cuts may reduce significantly the total number of nodes in the search tree.

The total number of nodes may be smaller in the search tree of a branch and cut tree as opposed to a branch and bound tree. Yet, in practice there may be a trade-off in terms of computational time. This is due to the fact that if many cuts are added at each node the reoptimization may be a lot slower.

To solve problem 5.1 I utilize branch and cut. Nevertheless, I did not add all possible C-G cuts at each branch. Instead, I used the fact that a cycle transversal gives a valid inequality for the convex hull of k -cores and only add this inequality. When the branching variable x_b is set to 0, one must only find a cycle transversal of the subgraph induced by $V \setminus x_b$. If x_b is set to 1, one must find a cycle transversal of the cycles containing x_b . Recall that in order to find a k -core, x_b must be part of a cycle. Hence, if x_b is not in any cycle, the branch associated with $x_b = 1$ is pruned. A useful trick to minimize the number of subproblems explored is to find a cycle transversal C' in the subgraph produced by the nonzero entries of the LP solution and fix $C' \subseteq C$, where C is a cycle transversal of the given graph.

Let us consider again Problem 5.1 for K_4 and $k = 2$ as formulated in Problem 5.4. As in the branch and bound example, let S be the feasible region. The initial LP value

is $\underline{z} = 1$. However, after adding an inequality associated with a cycle transversal, the improved LP value is $\underline{z} = 1.5$ with $x = (\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2})^t$. Figure 5.6 shows the partial branch and cut tree of Problem 5.4. Initially, it looks similar to the branch and bound tree of the same problem.

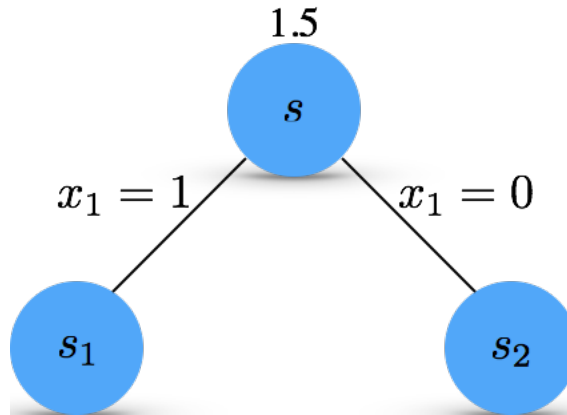


Figure 5.6 : Partial branch and cut tree for Problem 5.4.

I chose x_1 as the branching variable and S_1 as the node to be examined for the list of active problems, and add the cycle transversal inequality associated with $x_1 = 1$. The new LP solution is $\underline{z}_1 = 3$ and $\underline{x}^1 = (1, 1, 0, 1)^t$. S_1 lead us to an integral solution, the best feasible solution is updated $\bar{z} \leftarrow \min\{\bar{z}, 3\}$ and \underline{x}^1 is stored. S_1 is pruned by optimality as shown in Figure 5.7.

Now, consider S_2 as the node to be examined for the list of active problems, and add the cycle transversal inequality associated with $x_1 = 0$. The new LP solution is $\underline{z}_2 = 3$ and $\underline{x}^2 = (0, 1, 1, 1)^t$. S_2 gives an integral solution, but $\underline{z}_2 = \bar{z}$ and it is pruned by bound. The optimal solution is $\underline{x} = (1, 1, 0, 1)^t$ with objective function $z = 3$. In

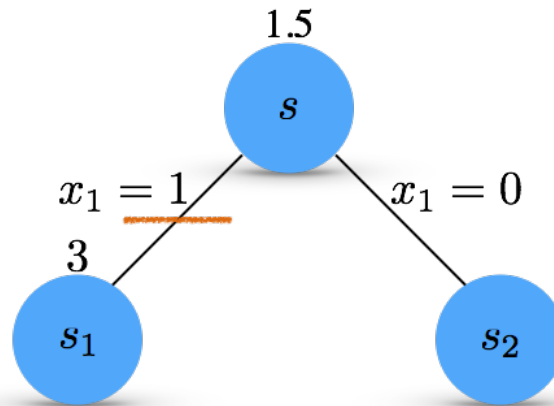


Figure 5.7 : The node associated with S_1 is pruned by bound.

Figure 5.8, the reader can see the full branch and cut search tree for Problem 5.4.

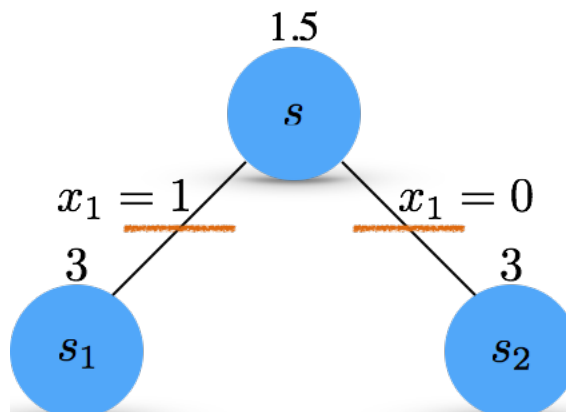


Figure 5.8 : Complete branch and cut tree for Problem 5.4.

5.6 Numerical Results

The approach discussed to solve Problem 5.1 was implemented using Matlab and tested in a MacBook Pro with an Intel Core i5 2.4 GHz Processor for $k = 2$ and 3. The results presented in this section were obtained by generating 100 Bernoulli random graphs for each of the probabilities used to generate edges. The cardinality of the vertex set is 100.

In the following plots, the reader will be guided through an analysis of the performance of my proposed branch and cut approach to solve Problem 5.1. I compare my approach with branch and bound as well as with the Matlab mixed-integer linear programming solver `intlinprog`. Branch and cut is denoted with circles and branch and bound with diamonds.

Figure 5.9 reports the time in seconds taken to solve Problem 5.1 for $k = 2$. It is clear that the branch and cut approach outperforms branch and bound. The time to solve the problem remains almost constant with the branch and cut approach. Figure 5.10 only displays the time in seconds taken to solve the desired problem with branch and cut approach. Although the average time changes depending on the density of the graph, the reader can observe that ninety percent of the time it takes a quarter of a second or less to solve Problem 5.1.

On Figure 5.11, one can explain the drastic decrease in time whenever cuts are added to the formulation. The number of nodes explored when solving the problem with branch and bound increases with the density of the graph. In contrast, the

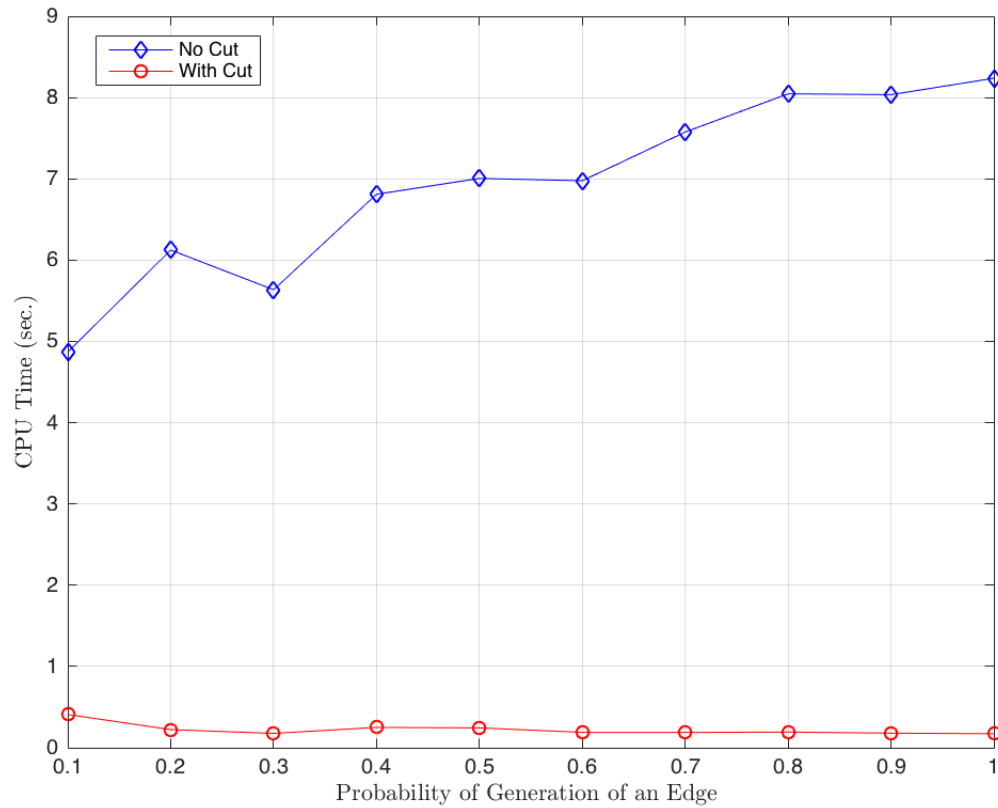


Figure 5.9 : Performance of branch and bound and branch and cut to solve Problem 5.1 for $k = 2$

branch and cut approach only solves between one and 9 subproblems and can be seen on Figure 5.12. Figure 5.13 emphasizes the advantage of introducing cuts for this instance of Problem 5.1. The bar plot shows the difference in the number of subproblems that must be solved using branch and bound minus the number of subproblems solved using branch and cut. On average, the number of nodes gained increases with the density of the graph.

Figure 5.14 reports the time in seconds taken to solve Problem 5.1 for $k = 3$.

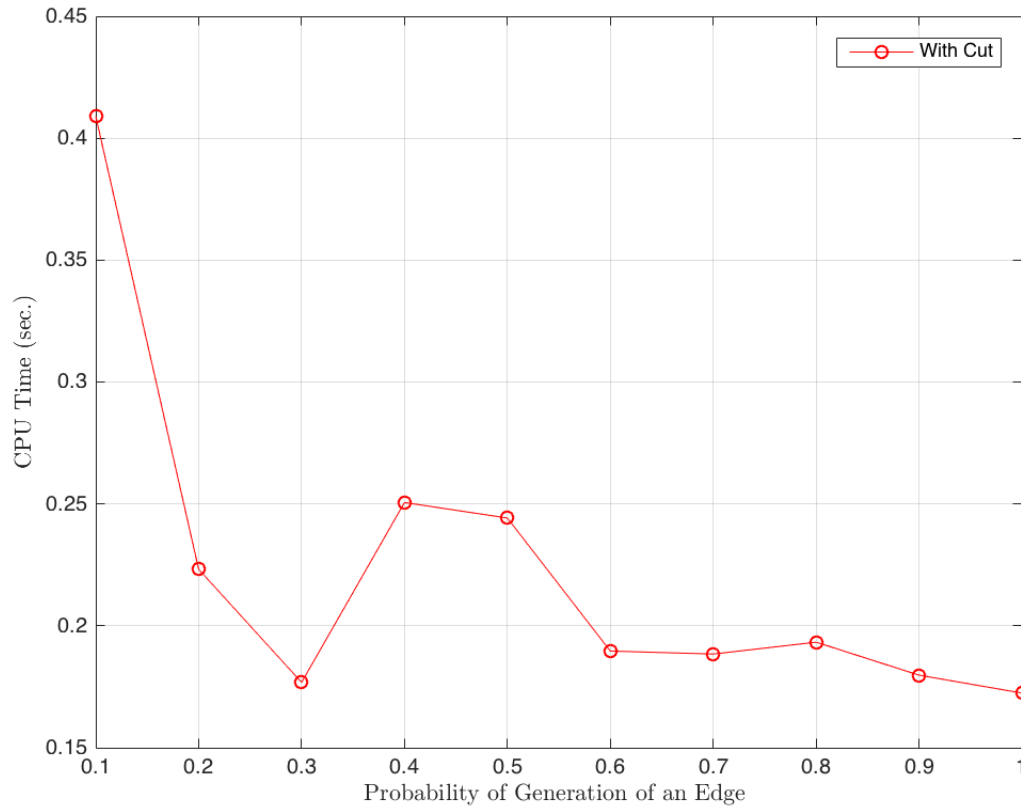


Figure 5.10 : Performance of branch and cut to solve Problem 5.1 for $k = 2$

Although our experiments show that if the graph is very sparse branch and bound is faster, ninety percent of the time is better to add the cut. Figure 5.15 displays the number of subproblems each method solved. As in the case for $k = 2$, overall the number of solved subproblems is much lower when solving with branch and cut. The number of subproblems solved by branch and bound increases with the density of the graph as opposed to branch and cut whose number of subproblems decreases or

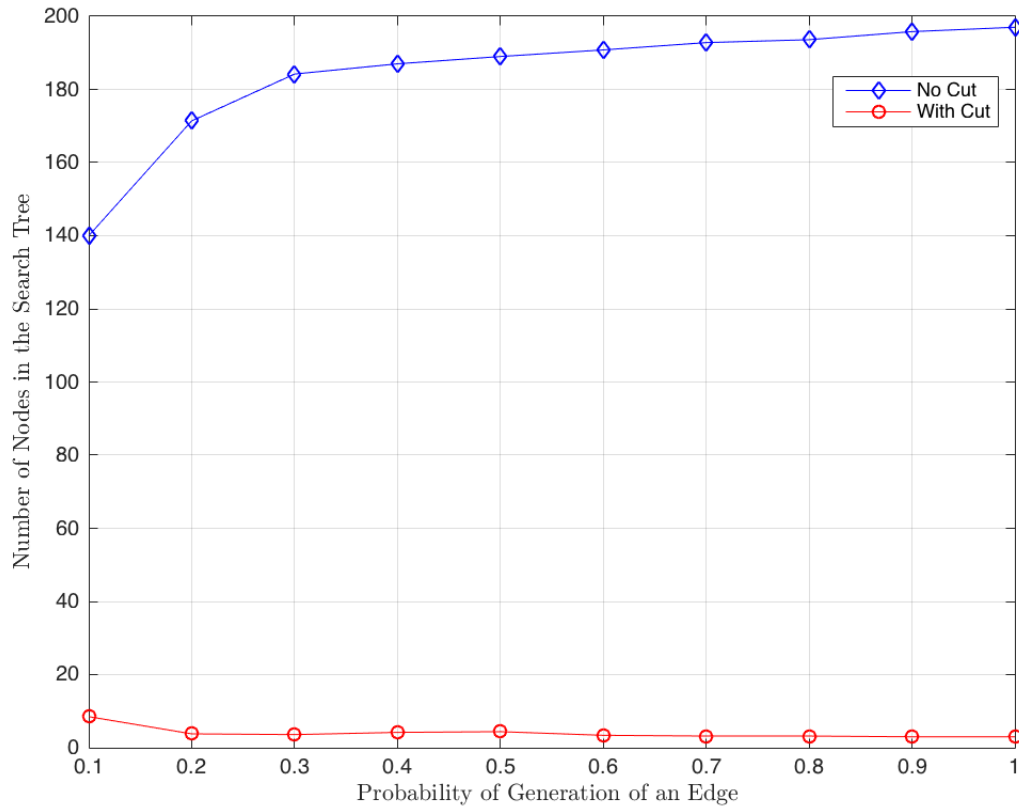


Figure 5.11 : Number of subproblems solved by branch and bound and branch and cut for $k = 2$.

remain constant, as can be seen in 5.16. The number of nodes gained by solving with branch and cut can be seen in Figure 5.17. Note that there is no bar for the graphs generated with probability equal to 0.1 due to the fact that branch and cut solves more subproblems for this particular instance.

Figure 5.18 compares the performance of the branch and cut approach, which I call my solver, versus Matlabs' mixed-integer linear programming solver Intlinprog. The average CPU time of one hundred test instances for value of number of nodes is

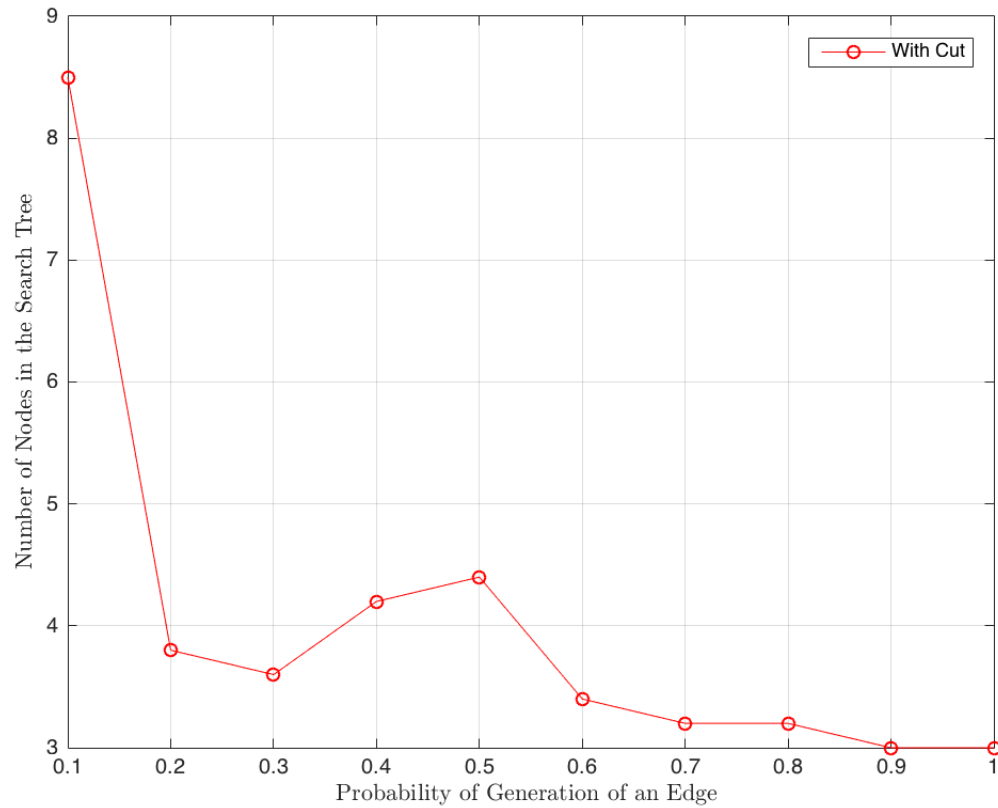


Figure 5.12 : Number of subproblems solved by branch and cut $k = 2$.

presented. The results for $k = 2$ are on the left hand side, and the right hand side corresponds to $k = 3$. The branch and cut approach is always faster than Intlinprog for $k = 2$. Moreover, the CPU time required to solve the problem as the graph gets larger grows at a slower rate than Intlinprog whose CPU time shows an exponential behavior.

In contrast to the case when $k = 2$, the branch and cut approach is slower than

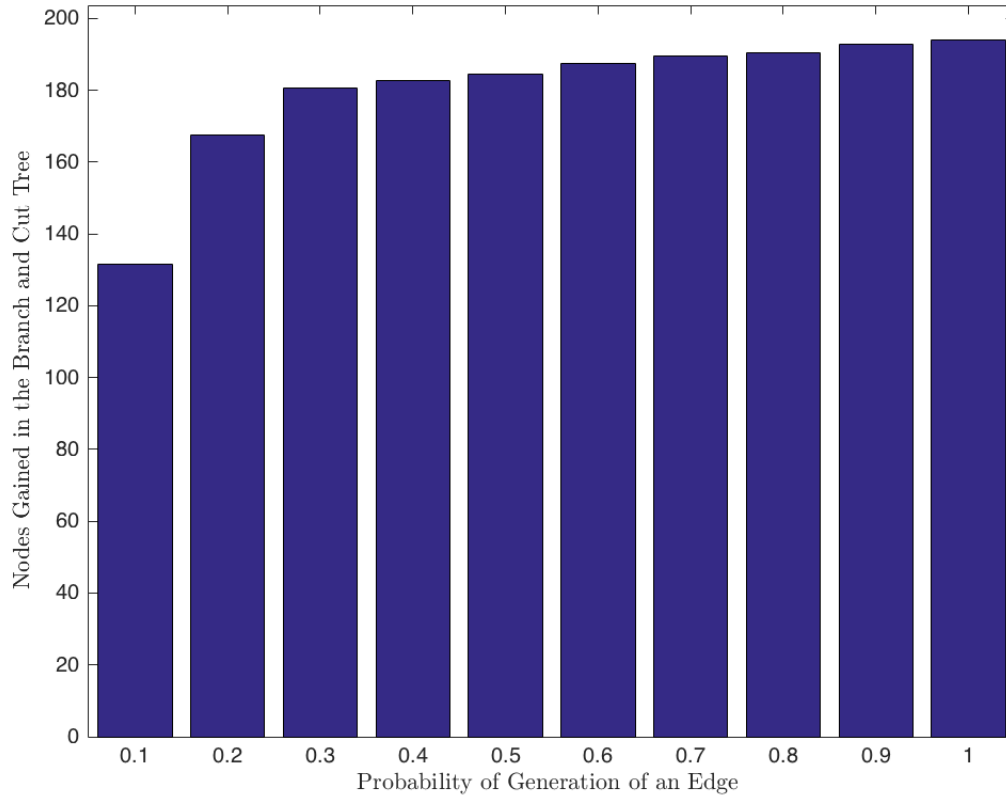


Figure 5.13 : Number of subproblems decrease when using branch and cut for $k = 2$.

Intlinprog when $k = 3$ and the graphs are sparse. However, the branch and cut approach is still recommended when $k = 3$ and graphs are generated with probability greater than or equal to 0.5. Similarly to the case when $k = 2$, the time difference between the two methods favors the branch and cut approach as the size of the graph increases.

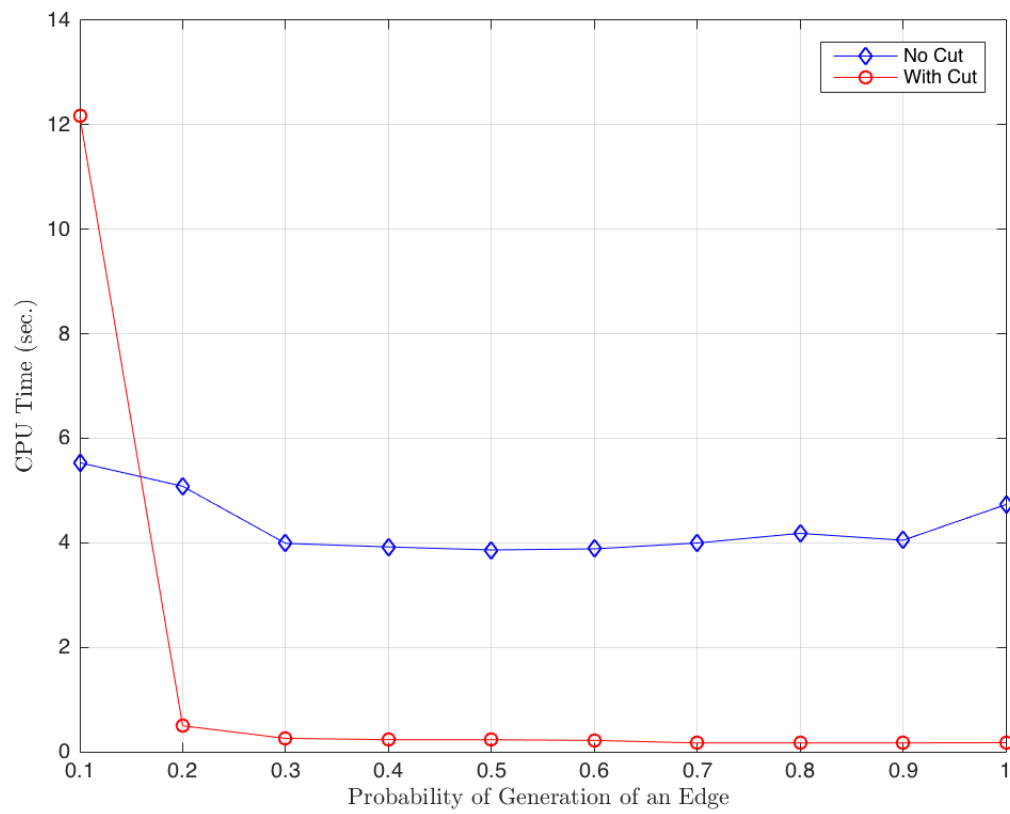


Figure 5.14 : Performance of branch and bound and branch and cut to solve Problem 5.1 for $k = 3$.

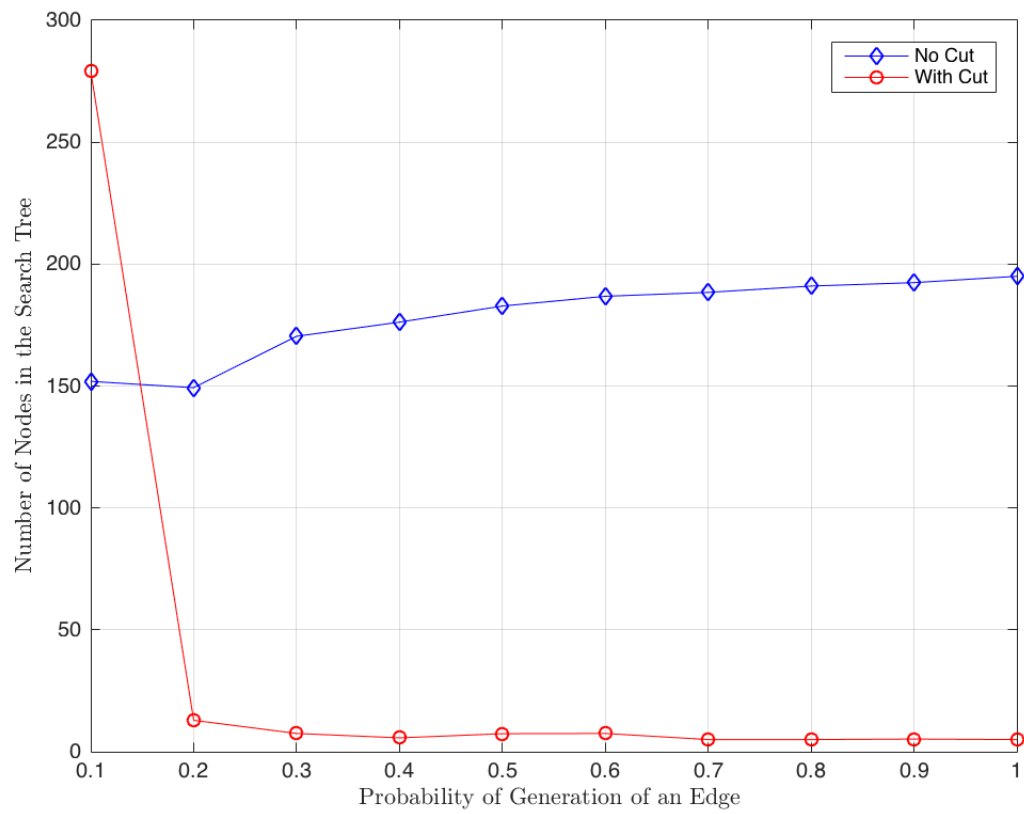


Figure 5.15 : Number of subproblems solved by branch and bound and branch and cut for $k = 3$.

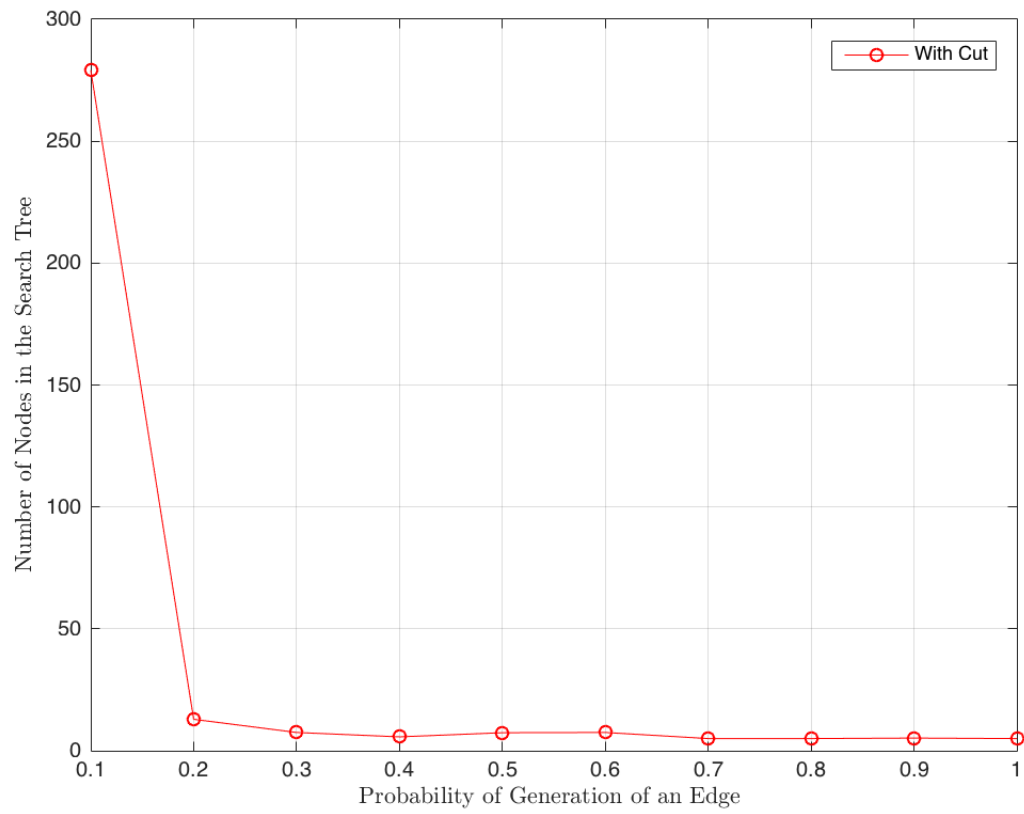


Figure 5.16 : Number of subproblems solved by branch and cut for $k = 3$.

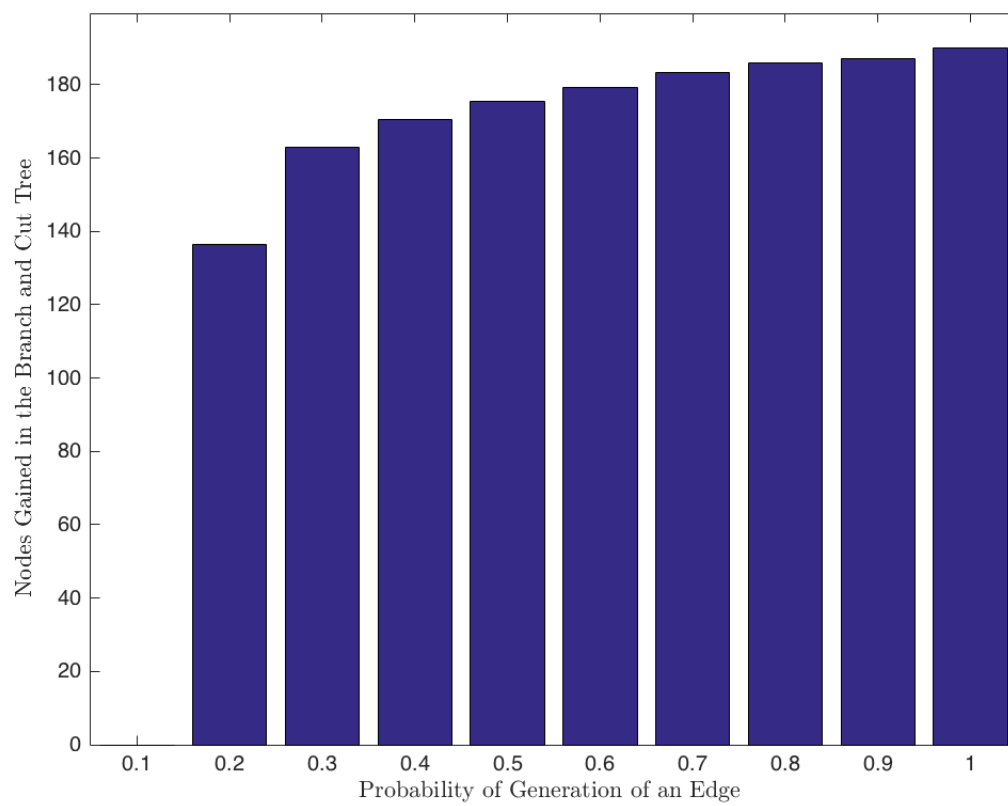


Figure 5.17 : Number of subproblems decrease when using branch and cut for $k = 3$.

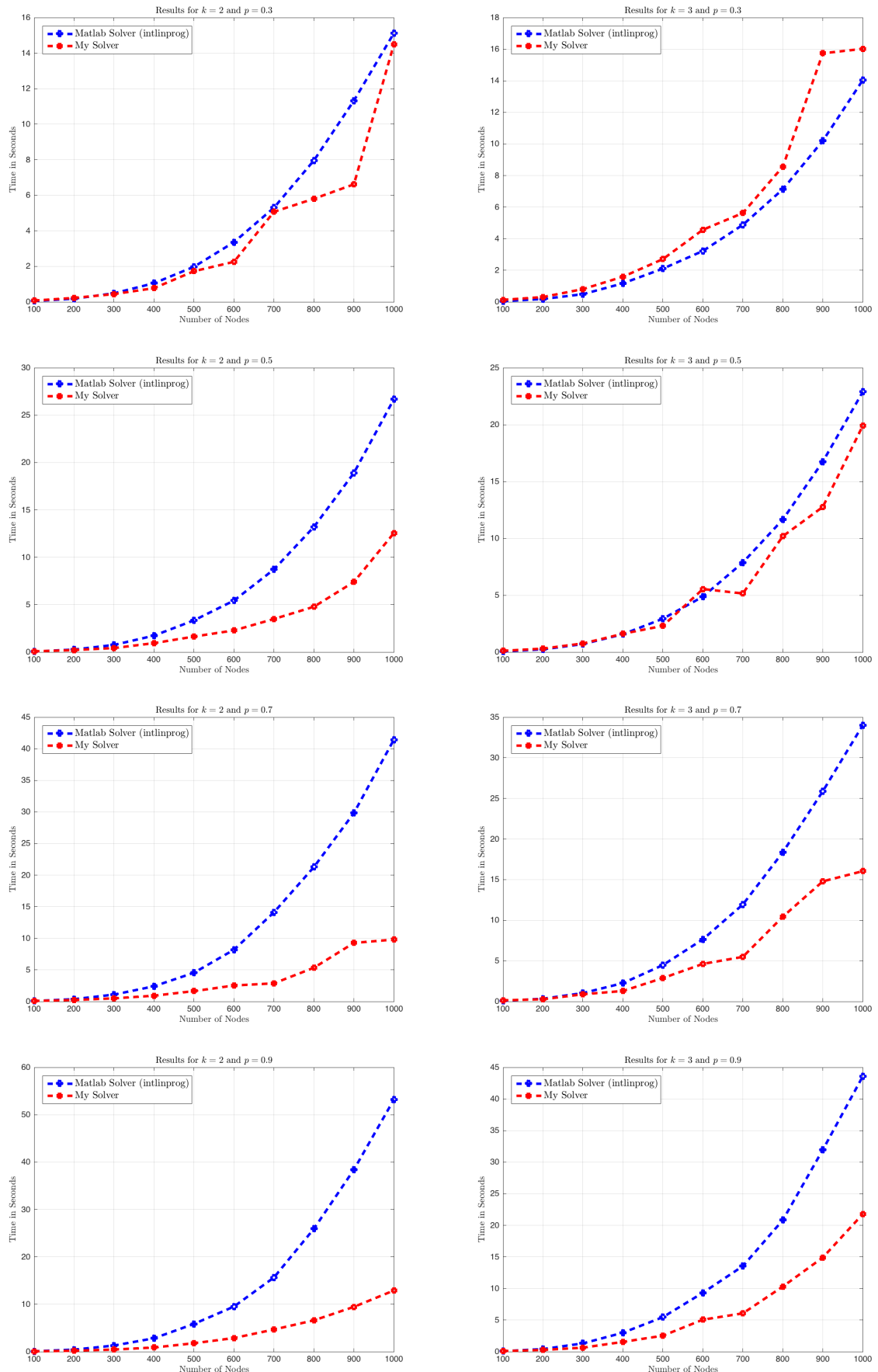


Figure 5.18 : Performance of branch and cut approach vs Matlab solver

Chapter 6

Conclusion

In this thesis, I presented three problems related to clique generalizations. They are the minimal k -core problem, the maximum weighted co-2-plex problem and the minimum k -core problem. My approach for solving these problems contributes to an area of research regarding clique generalizations. Some useful applications varying from sociological problems to biological models were mentioned throughout this thesis.

First, I discussed the minimal k -core problem and its relation to the study of associative memory. In particular, I discussed the link between the concept of cell assembly and the closure of a minimal k -core. I introduced a particular type of cell assembly called k -assembly. To accomplish the goal of this project which was to find all substructures within a network that must be excited in order to activate a k -assembly, I proposed a backtracking algorithm to solve the problem. The method is a modification of the Bron and Kerbosch algorithm for finding all cliques of an undirected graph. The results in the tested graphs offer insight in analyzing graph structure, help better understand how concepts are stored and show that important subgroups within a network overlap.

Secondly, I studied two possible ways to solve the MWC2P problem for {claw, bull}-free graphs. The two algorithms presented are solvable in polynomial time,

Minty's algorithm is extremely important for understanding both of them. One of them transforms the given graph to an auxiliary graph, then it applies Minty's algorithm; the other modifies Minty's algorithm to solve the problem directly. Through numerical experiments, I observed that it is better to use the modification of Minty's Algorithm to solve the problem in the original graph.

Finally, I presented a solution for the minimum k -core problem. In this work, I emphasize the fact that adding some additional inequalities to the LP relaxation of an IP may give a better solution. First, I set up the problem as an IP and relaxed it to obtain an LP. I showed that edge and cycle transversals of the graph give valid inequalities for the convex hull of k -cores. However, only the cycle transversal inequality proved to be useful in practice.

The three aforementioned problems add to the current line of research on clique and stable set generalizations. They can be used to model problems in a wide range of applications and different areas, including but not limited to biology and logistics.

Bibliography

- [1] E.A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1), 1973.
- [2] B. Balasundaram, S. Butenko, I. V. Hicks, and S. Sachdeva. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, January 2011.
- [3] Timothy Becker. A branch-and-cut method for solving the bilevel clique interdiction problem. Master’s thesis, RICE UNIVERSITY, 2015.
- [4] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.
- [5] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics Series. Springer, first edition, 2008.
- [6] Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *Science*, 323(5916):892–5, Feb 2009.
- [7] Valentino Braitenberg. Cell assemblies in the cerebral cortex. In *Theoretical approaches to complex systems*, pages 171–188. Springer, 1978.
- [8] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(575-577), 1973.
- [9] Sergiy Butenko and Wilbert E Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.
- [10] J. Kerbosch C. Bron and H.J. Schell. Finding cliques in an undirected graph. Technical report, Technological University of Eindhoven, The Netherlands, February 1972.
- [11] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3):5664–568, 2008.
- [12] A.M. Collins and E.F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

- [13] William Cook and Andre Rohe. Code for solving minimum-weight perfect matchings blossom4. <http://www.math.uwaterloo.ca/~bico/blossom4/>, 1999.
- [14] William Cook and Andre Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2):138–148, 1999.
- [15] S.J. Cox, J. Cavazos, K. Halani, and Z. Rubenstein. Cell assembly enumeration in random graphs. Technical report, Rice University, 2010.
- [16] Carina Curto and Vladimir Itskov. Cell groups reveal structure of stimulus space. *PLoS Comput Biol*, 4(10):e1000205, Oct 2008.
- [17] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.
- [18] Leon Festinger. The analysis of sociograms using matrix algebra. *Human Relations*, 2(153), 1949.
- [19] Elaine Forsythe and Leo Katz. A matrix approach to the analysis of sociometric data. *Sociometry*, 9(4):340–349, 1946.
- [20] M.R. Garey and D.S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, 1979.
- [21] D. O. Hebb. *The organization of behavior*. Wiley, 1949.
- [22] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.
- [23] Donald E Knuth. The art of computing programming. V, 2:198–213, 1968.
- [24] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. MIT press, 2001.
- [25] Longnian Lin, Remus Osan, Shy Shoham, Wenjun Jin, Wenqi Zuo, and Joe Z Tsien. Identification of network-level coding units for real-time representation of episodic experiences in the hippocampus. *Proc Natl Acad Sci U S A*, 102(17):6125–30, Apr 2005.
- [26] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2(3):253–267, 1972.
- [27] R. D. Luce and A.D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, Jun 1949.
- [28] Benjamin McClosky. *Independence systems and stable set relaxations*. PhD thesis, RICE UNIVERSITY, 2008.

- [29] Benjamin McClosky and Illya V Hicks. The co-2-plex polytope and integral systems. *SIAM Journal on Discrete Mathematics*, 23(3):1135–1148, 2009.
- [30] D A Miller and S W Zucker. Computing with self-excitatory cliques: A model and an application to hyperacuity-scale computation in visual cortex. *Neural Comput*, 11(1):21–66, Jan 1999.
- [31] George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980.
- [32] J. Moon and L. Mooser. On cliques in graphs. *Israel Journal in Mathematics*, 3(1):23–28, 1965.
- [33] Jacob L Moreno, Helen Hall Jennings, et al. *Who shall survive?* Nervous and mental disease publishing co., 1934.
- [34] Daishin Nakamura and Akihisa Tamura. A revision of minty’s algorithm for finding a maximum weight stable set of a claw-free graph. *Journal of the Operations Research Society of Japan*, 44(2):194–204, 2001.
- [35] Remus Oşan, Guifen Chen, Ruiben Feng, and Joe Z Tsien. Differential consolidation and pattern reverberations within episodic cell assemblies in the mouse hippocampus. *PLoS One*, 6(2):e16507, 2011.
- [36] G. Palm. On associative memory. *Biological Cybernetics*, 36:19–32, 1980.
- [37] G. Palm. Towards a theory of cell assemblies. *Biological Cybernetics*, 39:181–194, 1981.
- [38] Günther Palm, Andreas Knoblauch, Florian Hauser, and Almut Schüz. Cell assemblies in the cerebral cortex. *Biological cybernetics*, 108(5):559–572, 2014.
- [39] J Pattillo, N Youssef, and S Butenko. Clique relaxation models in network analysis: taxonomy, cohesiveness. Technical report, and optimization. Working paper, 2012.
- [40] David Picado-Muiño, Christian Borgelt, Denise Berger, George Gerstein, and Sonja Grün. Finding neural assemblies with frequent item set mining. *Frontiers in neuroinformatics*, 7, 2013.
- [41] Christina Prell. *Social network analysis: History, theory and methodology*. Sage, 2011.
- [42] Stephen R Proulx, Daniel E L Promislow, and Patrick C Phillips. Network thinking in ecology and evolution. *Trends Ecol Evol*, 20(6):345–53, Jun 2005.

- [43] Nicholas Rhodes, Peter Willett, Alain Calvet, James B Dunbar, and Christine Humblet. Clip: similarity searching of 3d databases using clique detection. *Journal of chemical information and computer sciences*, 43(2):443–448, 2003.
- [44] Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: uses and interpretations. *Neuroimage*, 52(3):1059–69, Sep 2010.
- [45] Najiba Sbihi. Algorithme de recherche d’un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980.
- [46] S.B. Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.
- [47] S.B. Seidman and B.L. Foster. A graph theoretic generalization of the clique concept. *The Journal of Mathematical Sociology*, 6:139–154, 1978.
- [48] Olaf Sporns. *Networks of the Brain*. MIT press, 2011.
- [49] Olaf Sporns and Rolf Kötter. Motifs in brain networks. *PLoS biology*, 2(11):e369, 2004.
- [50] M.T. Thai and P.M. Pardalos, editors. *Handbook of Optimization in Complex Networks: Communication and Social Networks*. Springer, 2012.
- [51] Minoru Tsukada, Natsuhiko Ichinose, Kazuyuki Aihara, Hiroyuki Ito, and Hiroshi Fujii. Dynamical cell assembly hypothesis - theoretical possibility of spatio-temporal coding in the cortex. *Neural Netw*, 9(8):1303–1350, Nov 1996.
- [52] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
- [53] Wasserman and Faust. *Social Network Analysis Methods and Applications*. Cambridge University Press, 1994.
- [54] Laurence A Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.
- [55] Cynthia I Wood and Illya V Hicks. The minimal k-core problem for modeling k-assemblies. *The Journal of Mathematical Neuroscience (JMN)*, 5(1):1–19, 2015.
- [56] R Kevin Wood. Bilevel network interdiction models: Formulations and solutions. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.
- [57] Tyler Young, S.J. Cox, and Illya V. Hicks. *The Art of the PFUG*, chapter Cell Assemblies: A Binary Integer Programming Problem. Number <http://cnx.org/contents/40d3a19a-b287-4e85-ba87-13b7b1606df3@34.1>. OpenStax CNX, 2013.