

RICE UNIVERSITY

# Cocircuits of Vector Matroids

by

**John David Arellano**

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

**Master of Arts**

APPROVED, THESIS COMMITTEE:



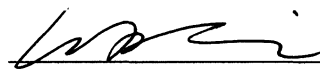
---

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



---

Richard A. Tapia  
Maxfield-Oshman Professor of  
Engineering  
University Professor of Computational  
and Applied Mathematics



---

Wotao Yin  
Assistant Professor of Computational and  
Applied Mathematics

Houston, Texas

August, 2011

## ABSTRACT

### Cocircuits of Vector Matroids

by

John David Arellano

In this thesis, I present a set covering problem (SCP) formulation of the matroid cogirth problem, finding the cardinality of the smallest cocircuit of a matroid. Addressing the matroid cogirth problem can lead to significantly enhancing the design process of sensor networks. The solution to the matroid cogirth problem provides the degree of redundancy of the corresponding sensor network, and allows for the evaluation of the quality of the network. I provide an introduction to matroids, and their relation to the degree of redundancy problem. I also discuss existing methods developed to solve the matroid cogirth problem and the SCP. Computational results are provided to validate a branch-and-cut algorithm that addresses the SCP formulation.

## Acknowledgments

I would like to thank my parents and family for the love and support throughout my graduate career. I would like to thank my advisor, Dr. Illya Hicks, and the rest of my committee for their guidance and support. I would also like to thank Dr. Maria Cristina Villalobos for her guidance and support during my undergraduate career. A thanks also goes to Nabor Reyna, Dr. Russell Carden and the rest of the AGEF family for their support.

# Contents

Abstract	ii
Acknowledgments	iii
List of Illustrations	vi
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Basic Terminology . . . . .	4
1.2.1 Matroid . . . . .	5
1.2.2 Dual Matroid . . . . .	7
1.2.3 Finding the Degree of Redundancy . . . . .	8
1.2.4 Set Covering Problem . . . . .	11
<b>2 Literature Review</b>	<b>13</b>
2.1 Solving the Cogirth Problem . . . . .	13
2.2 Solving the Set Covering Problem . . . . .	20
<b>3 Methods</b>	<b>26</b>
3.1 An Algorithm for the Set Covering Problem . . . . .	26
3.1.1 Generating Cutting Planes . . . . .	31
3.1.2 Branching . . . . .	34
3.1.3 Finding Feasible Solutions . . . . .	34
3.2 Rearranging Matrices into Bordered Block Diagonal Form . . . . .	35
<b>4 Computational Results</b>	<b>38</b>

<b>5 Conclusion</b>	<b>46</b>
<b>Bibliography</b>	<b>48</b>

## Illustrations

1.1	Sensor network example 1 . . . . .	3
1.2	Sensor network example 2 . . . . .	4
1.3	Matroid Example . . . . .	6
1.4	Hypergraph example for a matroid . . . . .	11
2.1	Bordered Block Diagonal Form . . . . .	15
3.1	Flow chart of basic Branch-and-Cut Algorithm. . . . .	29

## Nomenclature

$\mathcal{B}(M)$	The collection of bases of the matroid $M$
$\mathcal{C}(M)$	The collection of circuits of the matroid $M$
$\mathcal{B}^*(M)$	The collection of cobases of the matroid $M$
$\mathcal{C}^*(M)$	The collection of cocircuits of the matroid $M$
$M[Z]$	The matroid derived from the matrix $Z$ with $S$ representing the rows of $Z$ and $\mathcal{I}$ the collection of linear independent subsets of rows
$Col(H)$	The set of all possible linear combinations of column vectors of the matrix $H$
$G = (V, E)$	The graph $G$ with vertex set $V$ and edges $E$
$G_{hyp} = (V, E)$	The hypergraph $G$ with vertex set $V$ and hyperedges $E$
$H^T$	The transpose of the matrix $H$
$H_{(-d)}$	The matrix obtained from the $H$ after deleting $d$ rows of the matrix
$M = (S, \mathcal{I})$	The matroid $M$ with ground set $S$ and collection of independent sets $\mathcal{I}$
$null(H)$	$x \in \mathbb{R}^n$ such that $Hx = 0$ where $H$ is an $m \times n$ matrix
$r(H)$	The rank, or dimension of the column space, of the matrix $H$

$Row(H)$	The set of all possible linear combinations of row vectors of the matrix $H$
DoR	Degree of Redundancy



# Chapter 1

## Introduction

This thesis evaluates the accuracy and efficiency of five methods developed to solve the matroid cogirth problem. Solving this problem efficiently can aid in the process of designing sensor networks by providing the degree of redundancy of a given network. I present the matroid cogirth problem as an instance of the set covering problem and implement a branch-and-cut algorithm to solve it to optimality. Several test instances and numerical results are presented to validate the developed algorithm. Observations and possible avenues for future work are also discussed.

### 1.1 Motivation

Sensor networks play an important role in industry such as monitoring chemical plants [33]. Therefore, the ability to design a reliable sensor network is important. When designing a sensor network, the degree of redundancy of the network is a way to measure the reliability of the network. Measurements acquired from the network are *redundant* if they cannot only be obtained directly, but can also be obtained from another set of measurements. It is common practice to represent the relationship

between sensor measurements  $y$  and the system states  $u$  via a linear model:

$$y = Hu + \varepsilon,$$

where  $y$  is an  $n \times 1$  vector,  $u$  is a  $p \times 1$  vector,  $H$  is an  $n \times p$  ( $n > p$ ) matrix with rank  $p$ , and  $\varepsilon$  is an  $n \times 1$  vector representing noise accumulated by linearizing the system [23, 31]. The rank of the matrix refers to the dimension of the column space, the set of all possible linear combinations of its column vectors. In the remainder of this thesis including the computational results,  $\varepsilon$  is considered to be the zero vector (the system is noise free). The degree of redundancy (DoR) of the network refers to the number of sensors of the network that can fail while maintaining the ability to obtain all the measurements [24]. Therefore, a sensor network with a high DoR would be more reliable than a sensor network with a low DoR. By obtaining the DoR, it is possible to evaluate which sensor networks are more reliable than others.

Consider the example networks in Figures 1.1 and 1.2. In Figure 1.1,  $S_1, \dots, S_5$  represent sensors and  $M_1, \dots, M_5$  represent streams of information or measurements in the network measured by sensors to the right of them. Here, we assume that a measurement can be obtained by a sensor, or if a measurement to the right of it is known. In other words, if  $M_5$  is known, then  $M_4$  can be obtained. The same goes for  $M_1, M_2$  and  $M_3$ . If all the measurements are known, then this network has a degree of redundancy of 4 since each measurement can be obtained from the measurement directly to the right of it. In Figure 1.2, we now have six sensors, and six streams of information. In this example, a measurement can only be obtained from other

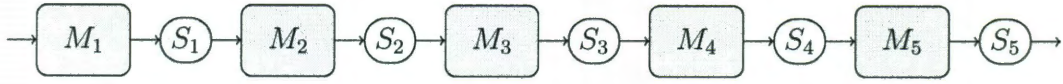


Figure 1.1 :  $S_1, \dots, S_5$  are sensors in a sensor network.  $M_1, \dots, M_5$  represent streams of information in the network. This example is adapted from an example given by Bagajewicz and Sanchez [3].

measurements if all the measurements directly to the right of it are known. For instance, both  $M_4$  and  $M_5$  are needed to obtain  $M_2$  without using sensor  $S_2$ . If we assume that all the measurements are known except for  $M_4$ , then the network has a DoR of 2 since  $M_3$  can be obtained from  $M_6$ , and  $M_1$  can be obtained from  $M_2$  and  $M_3$ . If all the measurements are known except for  $M_6$ , then we still have a DoR of 2 since  $M_2$  can be obtained from  $M_4$  and  $M_5$  and  $M_1$  is obtained as before. However, if  $M_4$  and  $M_6$  are not known, then the DoR is 1 since only  $M_1$  can be obtained from other known measurements. Different sensor network configurations may provide different matrices in the linear system described above. That is, the matrices may or may not have structure, in particular a bordered block diagonal form (BBDF) which is discussed in Chapter 2. However, the cases in which this structure does arise are still of great importance. In order to find the DoR of a network, we can describe a sensor failure using the matrix  $H$ .

Removing the  $k$ th row from  $H$  simulates a failure of the  $k$ th sensor in the network. Considering this, the degree of redundancy is defined as:

$$d^* = \min\{d - 1 : \text{there exists } H_{(-d)} \text{ such that } r(H_{(-d)}) < p\},$$

where  $H_{(-d)}$  represents a submatrix of  $H$  obtained by deleting  $d$  of the rows of  $H$ ,

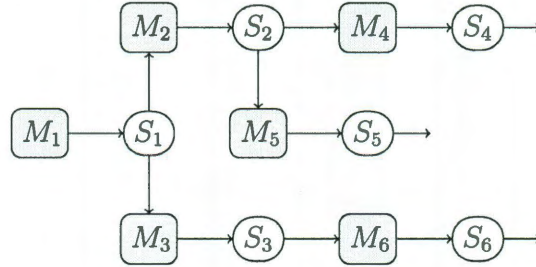


Figure 1.2 :  $S_1, \dots, S_6$  are sensors in a sensor network.  $M_1, \dots, M_6$  represent streams of information in the network. This example is adapted from an example given by Bagajewicz and Sanchez [3].

$r(H_{(-d)})$  is the rank of the resulting matrix and  $p$  is the rank of  $H$ . It is assumed that  $H$  is a matrix with full column rank. The DoR of the network can be found by finding the cogirth of the matroid obtained from the matrix  $H$ . The relationship between the DoR of a sensor network and the matroid cogirth problem will be discussed in the following section. In Chapter 2, I will discuss five existing methods that attempt to solve the cogirth problem accurately and efficiently.

## 1.2 Basic Terminology

This section describes some basic terminology and properties from matroid theory. This discussion is meant to familiarize the reader with concepts that will arise throughout this thesis. This discussion includes matroids, dual matroids, the cogirth problem and the set covering problem. The following definitions and properties are common in the literature. For a more comprehensive review of matroids, the reader is referred to Oxley [29]. For more introduction into the set covering problem, the

reader is referred to Nemhauser and Wolsey [28] or Wolsey [35]. Matroids were first introduced by Whitney [34] as a generalization of linear independence and linear dependence. Therefore, many of the terms and concepts may appear familiar.

### 1.2.1 Matroid

A *matroid*,  $M$ , consists of an ordered pair  $(S, \mathcal{I})$ .  $S$  is a finite set, and  $\mathcal{I}$  is a collection of subsets of  $S$  satisfying the following three properties.

(I1)  $\emptyset \in \mathcal{I}$ .

(I2) If  $I \in \mathcal{I}$  and  $J \subseteq I$ , then  $J \in \mathcal{I}$ .

(I3) If  $I_1, I_2 \in \mathcal{I}$  and  $|I_1| < |I_2|$ , then  $\exists e \in I_2 - I_1$  s.t.  $I_1 \cup e \in \mathcal{I}$ .

$S$  and the elements of  $\mathcal{I}$  are referred to as the *ground set* and *independent sets* of  $M$  respectively. All other subsets of  $S$  not in  $\mathcal{I}$ ,  $S - \mathcal{I}$ , are *dependent sets*.

Consider the real-valued matrix given in Figure 1.3. In this example, the ground set,  $\{1, 2, 3, 4, 5\}$ , corresponds to the rows of  $Z$ , and the independent sets correspond to sets of linearly independent rows of the matrix.

The matroid obtained from  $Z$  is called a *vector matroid* and is denoted by  $M[Z]$ . Note that a matroid can also be obtained by letting the ground set correspond to columns of a matrix and the independent sets correspond to sets of linearly independent columns. However, the matroid obtained by using the columns of the matrix is different than the matroid obtained by using the rows of the same matrix. For  $M[Z]$ , we have the following:

$$Z = \begin{matrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \end{matrix}$$

Figure 1.3 : Example of a matroid  $M = (S, \mathcal{I})$ . The ground set  $S$  corresponds to row indices and  $\mathcal{I}$  corresponds to sets of linearly independent rows.

$$S = \{1, 2, 3, 4, 5\}$$

$$\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{4\}, \{5\}, \{1, 2\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{4, 5\}\}$$

$$S - \mathcal{I} = \{\{3\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}, \{3, 5\}\} \cup \{X \subseteq S : |X| \geq 3\}$$

A *maximally independent set* of a matroid  $M$  is a set  $J \in \mathcal{I}$  such that  $J \cup x$  is dependent for all  $x \in S - J$ , and is referred to as a *basis*  $B$  of  $M$ . The collection of all bases of  $M$  is denoted as  $\mathcal{B}(M)$ . Consider rows 1 and 2 from  $Z$ . If you include any other row of  $Z$  with rows 1 and 2, then the set of rows becomes linearly dependent. Therefore,  $\{1, 2\}$  is a basis of  $M[Z]$ . The collection of bases of  $M[Z]$  is  $\{\{1, 2\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{4, 5\}\}$  and satisfy the following two properties. Observe that all the bases have the same cardinality.

$$(B1) \mathcal{B}(M) \neq \emptyset.$$

(B2) If  $B_1, B_2 \in \mathcal{B}(M)$  and  $x \in B_1 - B_2$ , then  $\exists y \in B_2 - B_1$  s.t.  $(B_1 - x) \cup y \in \mathcal{B}(M)$ .

A *minimally dependent set* of a matroid  $M$  is called a *circuit*  $C$  of  $M$ . A circuit  $C$  is a subset of  $S$  such that  $C - x \in \mathcal{I}$  for all  $x \in C$ . The collection of all circuits of  $M$  is denoted as  $\mathcal{C}(M)$ . The collection of circuits of  $M$  satisfy the following three properties.

(C1)  $\emptyset \notin \mathcal{C}(M)$ .

(C2) If  $C_1, C_2 \in \mathcal{C}(M)$ , and  $C_1 \subseteq C_2$ , then  $C_1 = C_2$ .

(C3) If  $C_1, C_2$  are distinct members of  $\mathcal{C}(M)$  and  $e \in C_1 \cap C_2$ , then  $\exists C_3 \in \mathcal{C}(M)$  s.t.  

$$C_3 \subseteq (C_1 \cup C_2) - e.$$

The cardinality of the smallest circuit is called the *girth* of the matroid. Referring back to the example, the collection of circuits of  $M [Z]$  is  $\{\{3\}, \{1, 4\}, \{1, 2, 5\}, \{2, 4, 5\}\}$  and the girth is 1.

### 1.2.2 Dual Matroid

Given a matroid  $M = (S, \mathcal{I})$ , we consider another matroid whose ground set is also  $S$ . Given  $S$ , define the following collection of subsets of  $S$ ,  $\{S - B : B \in \mathcal{B}(M)\}$ . This set, which will be denoted by  $\mathcal{B}^*(M)$ , is the set of bases of another matroid. This matroid denoted by  $M^*$  is called the *dual matroid* of  $M$ . Thus,  $\mathcal{B}(M^*) = \mathcal{B}^*(M)$  and  $(M^*)^* = M$ . If  $B \in \mathcal{B}^*(M)$ , it is called a *cobasis* of  $M$ . Referring back to

the example matrix  $Z$ , recall that  $\mathcal{B}(M[Z]) = \{\{1, 2\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{4, 5\}\}$ . Therefore,  $\mathcal{B}^*(M[Z]) = \{\{3, 4, 5\}, \{2, 3, 4\}, \{1, 3, 5\}, \{1, 3, 4\}, \{1, 2, 3\}\}$ .

Similarly,  $\mathcal{C}(M^*) = \mathcal{C}^*(M)$ , and if  $C \in \mathcal{C}^*(M)$ , it is called a *cocircuit* of  $M$ . The cardinality of the smallest cocircuit is called the *cogirth* of the matroid. Referring back to the example,  $\mathcal{C}^*(M[Z]) = \{\{1, 2, 4\}, \{1, 2, 5\}, \{2, 4, 5\}, \{1, 4, 5\}\}$  and the cogirth is 3. It should be noted that the bases and circuits of  $M^*$  are the cobases and cocircuits of  $M$  respectively and vice versa. The next sections discuss how a solution for the set covering problem provides a solution for the degree of redundancy.

### 1.2.3 Finding the Degree of Redundancy

The degree of redundancy (DoR) of a sensor network provides a measure for the reliability of a given network. The ability to design reliable networks that are cost efficient is important. Therefore, it is important to find the DoR of a sensor network. The remainder of this subsection describes the relationship of the degree of redundancy (DoR) of a sensor network and the cogirth of a matroid.

Although the mathematical definition of the DoR of a sensor network provides a simple algorithm to find the DoR, the algorithm itself is by no means practical. Therefore, it is necessary to take a different approach towards the problem. Recall the linear system,  $y = Hu + \varepsilon$ , which is a standard description of the sensor network where  $H$  is an  $n \times p$  matrix,  $n > p$ . Define  $R = \{1, \dots, n\}$  to be the set of row indices,  $SR$  to be the subset of rows of  $H$  removed, and  $d$  to be the cardinality of  $SR$ . The



cardinality of the smallest subset,  $\widehat{SR}$ , of rows such that the rank of the resulting matrix is one less than the rank of  $H$  is what needs to be computed. The DoR is  $\hat{d} - 1$ , where  $\hat{d} = |\widehat{SR}|$ , the cardinality of  $\widehat{SR}$ . Note that one could consider  $H^T$  and remove sets of columns. Define  $RowS = \{r_1, r_2, \dots, r_l\}$  where  $l$  is the number of distinct bases of the row space of  $H$  and  $r_j$  is a distinct basis for  $j \in \{1, \dots, l\}$ . Therefore,  $|\widehat{SR}| = \min\{|J| : J \subseteq R, J \cap r_j \neq \emptyset \forall r_j \in RowS\}$ . In other words, the rows of the resulting matrix  $H_{(-d)}$  do not span the row space of  $H$  because the removed rows intersect every basis of the row space of  $H$ . Therefore, the rank of  $H_{(-d)}$  must be less than the rank of  $H$ . Recall the example matrix from Figure 1.3. The row space of  $Z$  is  $\mathbb{R}^2$ . By removing rows  $\{1, 2, 4\}$ , which is a cocircuit of  $M[Z]$ , the resulting matrix does not span  $\mathbb{R}^2$ . Switching back to the matrix  $H$ , we can obtain a vector matroid  $M[H]$  from  $H$ , where the distinct bases of  $M[H]$  correspond to distinct bases of the row space of  $H$ . Based on the observation that  $\{1, 2, 4\}$  is a cocircuit of  $M[Z]$ , one might think that removing rows from  $H$  that correspond to a cocircuit will reduce the rank of the resulting matrix. Upon this belief, in order to find the DoR, one can consider finding the cogirth of vector matroid  $M[H]$ . The following paragraphs build upon this observation, and describe why the cogirth of a vector matroid provides the DoR of the corresponding sensor network.

Recall that for a matroid  $M = (S, \mathcal{I})$ , the cobases are just the complement of the bases with respect to  $S$ . Since a basis,  $B$ , is maximally independent,  $B \cup x$  contains a circuit for all  $x \in S - B$ . This might lead one to wonder if a circuit has a nonempty

intersection with every basis in  $\mathcal{B}(M)$ . Oxley [29] shows that this is in fact the case. A *hypergraph*,  $G_{hyp} = (V, E)$ , is a generalization of a graph where members of  $E$  can connect two or more members of  $V$ . Define  $G_{hyp,H} = (V, E)$ , to be a hypergraph where  $V$  corresponds to the set of row indices of  $H$  and a hyperedge  $f \in E$  corresponds to a basis of  $M[H]$ . A *transversal* of  $G_{hyp}$  is a subset  $U$  of  $V$  such that every  $f \in E$  is adjacent to at least one member of  $U$ . Observe that since  $E$  and  $U$  correspond to the set of bases and a subset of row indices respectively and  $U$  touches every hyperedge,  $U$  corresponds to a cocircuit of  $M[H]$ . Therefore, if we want to find the cogirth of  $M[H]$ , we need to find the smallest transversal of the  $G_{hyp,H}$ . Similarly, if we wanted to find the girth of  $M[H]$ , we would consider a hypergraph with members of  $E$  corresponding to cobases of  $M[H]$ .

To provide a more illustrative example, refer back to the matrix  $Z$ . We can find the girth of  $M[Z]$  by considering the above mentioned hypergraph with members of  $E$  corresponding to cobases of  $M[Z]$ . Recall the collection of cobases of  $M[Z]$ ,  $\mathcal{B}^*(M[Z]) = \{\{3, 4, 5\}, \{2, 3, 4\}, \{1, 3, 5\}, \{1, 3, 4\}, \{1, 2, 3\}\}$ . The corresponding hypergraph can be seen in Figure 1.4. Notice that every member of  $\mathcal{C}(M) = \{\{3\}, \{1, 4\}, \{1, 2, 5\}, \{2, 4, 5\}\}$  is a transversal of the hypergraph and the smallest transversal is  $\{3\}$ . Therefore, the girth of the  $M[Z]$  is 1. In order to find the minimum transversal, we can turn to discrete optimization and model the problem as an integer program. A description of the set covering problem (SCP), and how it relates to the matroid cogirth problem is provided in Section 1.2.4.

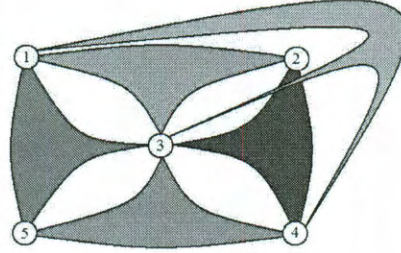


Figure 1.4 : This figure shows the hypergraph  $G_{hyp} = (V, E)$  where  $V$  corresponds to row indices of the matrix  $Z$  and members of  $E$  correspond to cobases of  $M[Z]$ .

### 1.2.4 Set Covering Problem

It is not uncommon to construct linear and integer programs for graph theory problems. This section shows how the cogirth problem can be described by an integer program. Let  $N = \{1, \dots, n\}$  and  $K = \{1, \dots, k\}$  for  $n, k \in \mathbb{Z}^+$ . Let  $N_1, N_2, \dots, N_k$  be a given collection of subsets of  $N$ . Each  $N_i$  is given a weight  $c_i$ .  $D$  which is a subset of  $K$  is called a *cover* of  $N$  if  $\bigcup_{i \in D} N_i = N$ . The weight of a cover  $D$  is  $\sum_{i \in D} c_i$ . The *set covering problem* (SCP) is  $\min \{c^T x \mid Ax \geq \mathbf{1}, x \text{ binary}\}$  where  $A$  is an incidence matrix of  $\{N_i \mid i \in K\}$ . The entries of  $A$ ,  $a_{ij}$ , are 1 if  $j \in N_i$  and 0 otherwise for all  $j \in N$ . Further introduction of the SCP can be found in [10], [28] and [35].

Recall the cogirth problem can be solved by addressing a minimum transversal problem. The problem of finding the minimum transversal can be seen as a SCP. In order to find the cogirth of  $M[H]$ , let  $N = \{1, \dots, n\}$ , where  $n$  is the number of rows of  $H$  and let  $K = \{1, \dots, k\}$  where  $k$  is the number of distinct bases of  $M[H]$ . Let  $N_i$  correspond to a distinct basis of  $M[H]$  with  $c_i = 1$  for each  $i \in K$ . For the incidence matrix  $A$ ,  $a_{ij}$  will be 1 if row  $j$  is a member of basis  $N_i$  and 0 otherwise. Therefore, the

problem now becomes  $\min \{\mathbf{1}^T x \mid Ax \geq \mathbf{1}, x \text{ binary}\}$ . The constraints demand that any feasible solution intersect every basis of  $M[H]$ , which is exactly what is required. In order to find the girth, let  $k$  be the number of distinct cobases of  $M[H]$ , and let  $N_i$  correspond to a distinct cobasis of  $M[H]$ . Therefore, a solution to the SCP provides a solution to the cogirth problem for vector matroids, which in turn provides the DoR of the corresponding sensor network. Since both problems are NP-hard, note that in general, it is possible to take a set covering problem, and model it as a cogirth problem for a vector matroid so that a solution to the cogirth problem provides a solution to the set covering problem. However, describing a clear path to do so is not discussed in this thesis. Further discussion of the SCP as well as methods devised to solve the SCP will be given in Chapter 2.

The remainder of this thesis focuses on solving the matroid cogirth problem by addressing the set covering problem. A review of existing methods developed to solve the matroid cogirth problem and the set covering problem is presented in Chapter 2. In Chapter 3, a brief discussion of the branch-and-cut algorithm is given. An implementation of the algorithm specific to the set covering problem formulated for the matroid cogirth problem is also provided. Computational results obtained from the implementation of the algorithm on some test instances will be presented in Chapter 4. Chapter 5 will focus on observations from the computational results, some areas where there is room for improvement, and some future work.

## Chapter 2

### Literature Review

#### 2.1 Solving the Cogirth Problem

Finding the cogirth of a general vector matroid is equivalent to finding the minimum transversal of a hypergraph, which can be solved as a set covering problem. As an optimization problem, the set covering problem is NP-hard [21]. Therefore a polynomial time algorithm to find the cogirth of a general vector matroid does not likely exist. However, there have been several attempts to develop an algorithm or heuristic that solves the matroid cogirth problem accurately and/or efficiently. The first algorithm, which is fairly obvious and simple in nature, comes from the definition of the DoR. Recall that for a sensor network with model matrix  $H$ , the DoR is defined as the cardinality of the smallest set of rows whose removal from  $H$  reduces the rank of the resulting matrix  $H_{(-d)}$  where  $d$  is the number of sets removed. Exhaustive rank testing is a brute force technique. As the definition of the DoR suggests, the idea is to iteratively remove sets of rows from  $H$  until the rank of  $H_{(-d)}$  is one less than the rank of  $H$ . Although the algorithm is easy to understand, the number of possible combinations of sets of rows that are to be removed is  $n$  choose  $d$  where  $n$  is the number of rows of  $H$ . As  $d$  increases, so does the number of possible combina-

tions. The singular value decomposition (SVD) is typically used to find the rank of matrices. However, since the computational time needed to compute the SVD of a matrix is dependent on the number of rows and columns of the matrix, the SVD can be computationally expensive on large submatrices. For practical system matrices, as  $d$  increases, the number of submatrices considered increases. In turn, the algorithm becomes computationally expensive as the number of submatrices increases. Therefore, it would not be advantageous nor practical to do this exhaustive rank testing as  $d$  and the size of the matrices increase.

Boros et al. [22] presented an algorithm to enumerate all the circuits of a matroid. The algorithm is based on property (C3) from Chapter 1. Let  $M = (S, \mathcal{I})$  be a matroid with ground set  $S$ . Given such a matroid, a basis,  $B \subset S$ , of  $M$ , and an element  $x \in S \setminus B$ ,  $B \cup x$  contains a circuit  $C$  of  $M$ . Using the same  $B$  and all  $x \in S \setminus B$ , a collection of initial circuits is obtained. Using (C3), it is concluded that all the circuits of the matroid have been enumerated or a new circuit is found, and added to the collection. These steps are repeated until all the circuits of the matroid have been enumerated. In the case of vector matroids, finding a basis, finding circuits and using (C3) as a check is not difficult. As the algorithm is guaranteed to enumerate all the circuits of the matroid, it proves to be quite accurate. However, as the number of circuits increases, the number of pairwise comparisons using (C3) increases. The number of circuits may not be known, and the algorithm also requires the enumeration of all the circuits of the matroid before it terminates. This enumeration is unnecessary

$$\begin{pmatrix} A_1 & & & & & \\ & A_2 & & & & \\ & & \ddots & & & \\ & & & & & \\ & & & & A_n & \\ P_1 & P_2 & \cdots & & & P_n \end{pmatrix}$$

Figure 2.1 : Bordered block diagonal form [13].

since only the ones of smallest cardinality are needed. Having to generate all the circuits can become quite taxing, especially if there are a large number of them. It would be advantageous to avoid this enumeration.

As an alternative, Cho et al. [13] presented a branch-and-decompose algorithm. The theory behind this method stems from the connectivity of the matroid constructed from the given matrix. It is assumed that the matroid maintains a degree of disconnectedness. For a more detailed discussion of matroid connectivity and the properties used to develop this algorithm, refer to Cho et al. [13]. The most important observation is that for a matrix  $T$ , the cogirth of  $M[T]$  can be found by looking at submatrices of  $T$ . The matrix  $T$  is transformed into bordered block diagonal form (BBDF), which can be seen in Figure 2.1, with a border,  $P$ , of rows. The BBDF matrix is a rearrangement of the columns and rows of  $T$ . While  $d$  is small, the exhaustive rank test is used on the entire matrix. That is, while  $d < \left(\frac{n_b}{n_b-1}\right)|P| - 1$

where  $n_b$  is the number of blocks and  $|P|$  is the number of rows in the border, the exhaustive rank testing is used. Once  $d$  reaches this bound, the algorithm intelligently chooses rows and columns of the matrix to create submatrices, and implements the exhaustive rank test on each submatrix to compute the cogirth of the original matrix. This method depends on the structure of the matroid and inherently on the matrix. Although it has the ability to provide nice results as presented by Cho et al. [13], it is not guaranteed to do so for all matrices with the desired structure. In particular, as the number of rows in  $P$  increases, so does the number of exhaustive rank tests in the initial stage of the algorithm. This algorithm, although very clever, runs the risk of being very inefficient if the matrix and in turn the matroid does not have the structure illustrated by the figure above with  $|P|$  small. Other methods have made the attempt to solve the problem for general vector matroids.

In contrast to the methods discussed above, Kianfar et al. [23] approached the problem from an optimization standpoint. They presented a 0-1 mixed integer program (MIP) to solve the cogirth problem. Given an  $n \times p$  matrix  $H$ , with  $n > p$ , the formulation attempts to find a nonzero vector  $x \in \text{null}(H)$ , the null space of  $H$ , that minimizes the number of nonzero inner products  $h_i x$  for  $i = 1, \dots, n$ , where  $h_i$  is a row of  $H$ . The assumptions are that  $\|h_i\|_1 = 1$  for each row of  $H$  and that  $H$  has full column rank since  $n > p$ . The following is the 0-1 MIP formulation:

$$\begin{aligned} \min \quad & \sum_{i=1}^n q_i \\ \text{s.t.} \quad & -q_i \leq \sum_{j=1}^p h_{ij} x_j \leq q_i, \quad i = 1, \dots, n \end{aligned}$$



$$-1 + 2z_j \leq x_j \leq 1, \quad j = 1, \dots, p$$

$$\sum_{i=1}^p z_i = 1$$

where  $x_j$  is a real variable for all  $j$  and  $z_i, z_j$  are binary variables for all  $i$  and  $j$ . The first set of constraints correspond to  $x$  being in the null space of  $H$ . The second and third set of constraints ensure that the trivial solution  $x = 0$  is not chosen. If the matrix is not full rank then another set of constraints must be added to ensure that  $x$  is in the row space of  $H$ . The formulation is then solved using the MIP solver CPLEX [1]. Computational comparisons to the exhaustive rank test and Cho's branch-and-decompose algorithm are presented in Table 1 in [23]. In some cases presented, the algorithm proposed by Cho et al. performs better, but the MIP formulation is shown to do better in most instances, especially as the size of the matrix increases. Upper and lower bounds for the degree of redundancy can still be acquired when optimality is not achieved [23]. The 0-1 MIP formulation reported is an initial approach, and the investigation is ongoing. The formulation sounds promising, especially if strong valid inequalities can be added to strengthen the formulation.

In order to find the cogirth of a matroid, Govindaraj [17] concentrates on the circuits of the dual matroid,  $M^*$ , which are the cocircuits of  $M$ . Rather than enumerate all cocircuits, an  $\ell^0$ -norm minimization formulation is used to find the smallest cocircuits containing each element in the ground set  $S$ . For a given matrix  $A$  and a vector  $b$ , the following formulation will provide the minimum number of columns of  $A$  needed to obtain  $b$ .

$$\begin{aligned} & \min \|x\|_0 \\ & \text{s.t. } Ax = b \end{aligned}$$

It should be noted that the  $\ell^0$ -norm is not actually a norm. It refers to the number of nonzeros of a vector. Rather than solve the  $\ell^0$ -norm minimization problem, an  $\ell^1$ -norm minimization approximation is used. In other words, replace  $\|x\|_0$  with  $\|x\|_1$ . Here,  $\|x\|_1$  refers to the 1-norm of a real-valued vector. Recall the matrix  $H$  from the linear system. This heuristic would be performed on  $H^T$ . The full algorithm consists of computing the matrix representation of the dual matroid, which is discussed in Oxley [29], and then solving a sequence of  $\ell^1$ -norm minimization problems. For each column,  $H_j$ ,  $j \in \{1, \dots, n\}$ , of the  $H^T$ , let  $b = H_j$  and let  $A$  be the matrix  $H^T$  excluding  $H_j$  and then solve the  $\ell^1$  approximation. Computational results reported by Govindaraj [17] indicate that the algorithm is efficient, but because the solutions are approximated, the algorithm is not always guaranteed to provide exact solutions. That being said, the inaccurate solutions still provide a good approximation to the true solution and the cogirth. The algorithm provides a viable option to give an approximation to the cogirth problem. The algorithm provided by Govindaraj is based on a technique to find an approximation to a problem that arises in compressive sensing. This problem is equivalent to finding the girth of a vector matroid. Along those lines one could also consider using other algorithms that address the compressive sensing problem such as the Basis Pursuit algorithm [12] and the Orthogonal Matching Algorithm [30],[32] as possible approximation algorithms to find the cogirth of a

vector matroid. In turn, the algorithms discussed above as well as the one presented in this paper could be considered to address the problem that arises in compressive sensing.

Another possible option is given in a more general setting by Moreno Centeno [26]. Moreno Centeno provides an algorithm to solve an implicit hitting set problem (IHSP). According to Moreno Centeno, the explicit hitting set problem (EHSP) is “identical to the classic weighted set cover problem, except that the roles of sets and elements have been interchanged” [26]. The goal is to find a hitting set that intersects all the “circuits.” Here, hitting sets and circuits are analogous to cocircuits and bases of the matroid  $M$ . The IHSP is like the EHSP except that the circuits are not explicitly known for the problem. As such, the circuits need to be generated using a separation oracle. As these circuits are generated, a hitting set is also generated to satisfy the current list of circuits. Although a list of circuits is generated, the algorithm tries to avoid solving the EHSP with the list of circuits if possible. When the EHSP is solved, it is done also using CPLEX [1]. The separation oracle used to generate circuits is specific to the problem being solved by Moreno Centeno [26]. Computational results are provided for this specific problem. Like Govindaraj [17], the method solves most reported test instances accurately and efficiently. However, there are some reported cases where the algorithm is not able to find the exact solution [26]. The algorithm may be a viable option to address the cogirth problem, but it depends on whether or not a separation oracle for the cogirth problem can be

developed to generate bases of a matroid efficiently. An investigation into adapting this method for the cogirth problem is needed.

## 2.2 Solving the Set Covering Problem

Recall the set covering problem. Let  $N = \{1, \dots, n\}$  and  $K = \{1, \dots, k\}$  for  $n, k \in \mathbb{Z}^+$ . Let  $N_1, N_2, \dots, N_k$  be a given collection of subsets of  $N$ . Each  $N_i$  is given a weight  $c_i$ .  $D$  which is a subset of  $K$  is called a *cover* of  $N$  if  $\bigcup_{i \in D} N_i = N$ . The weight of a cover  $D$  is  $\sum_{i \in D} c_i$ . The *set covering problem* (SCP) is  $\min \{c^T x \mid Ax \geq \mathbf{1}, x \text{ binary}\}$  where  $A$  is an incidence matrix of  $\{N_i \mid i \in K\}$ . The entries of  $A$ ,  $a_{ij}$ , are 1 if  $j \in N_i$  and 0 otherwise for all  $j \in N$ . The decision version of the SCP was proved to be NP-complete [21]. Posed as an optimization problem, the SCP is NP-hard. The SCP is a well-studied problem. There are a variety of known methods that have been developed to address the set covering problem. These methods include greedy heuristics, genetic algorithms, and other methods that implement cutting plane methods and Lagrangian heuristics.

Greedy heuristics that can find near optimal solutions are discussed by Chvátal [15] and Johnson [20]. Greedy heuristics work by trying to gain as much as possible at each given step of the procedure. Given the system  $Ax \geq \mathbf{1}$ , with  $A$  an  $m \times n$  matrix, the greedy algorithm would consist of the following steps.

Set  $Cov = \emptyset$ ,  $Uncov = \{1, \dots, m\}$ ,  $Sol = \emptyset$

**while**  $Uncov \neq \emptyset$  **do**

Choose  $j$ th column of  $A$  and  $Sol = Sol \cup j$  based on weights given to the columns of  $A$

**for**  $i = 1 \rightarrow m$  **do**

**if**  $a_{ij} = 1$  **then**

$Cov = Cov \cup i, Uncov = Uncov - i$

**end if**

**end for**

**end while**

$Sol$  is a solution to the SCP

The weights given to the columns of  $A$  can be obtained in many ways. For instance, let  $CR_j$  be the subset of rows of  $A$  such that  $a_{ij}$  is 1 for the  $j$ th column of  $A$ . Then we can let the weight of the  $j$ th column,  $w_j$ , be  $|CR_j|/c_j$ , where  $c_j$  is the cost of the  $j$ th column of  $A$ . In the SCP discussed in Chapter 2,  $w_j = |CR_j|$  for all columns of  $A$  since  $c = \mathbf{1}$ . There have also been attempts to improve upon this initial greedy algorithm. Marchiori and Steenbeek [25] offer what they call an iterated approximation algorithm (ITEG). Their algorithm basically starts by finding a cover using the procedure listed above with the weight of the  $j$ th column  $w_j = |CR_j|$ . A function they call *Enhanced Greedy* tries to improve the cover by adding or removing columns from the original cover using criteria which is discussed in more detail in [25]. The following procedure is performed for a specified number of iterations. Given a best solution  $S_{best}$ , a subset of  $S_{best}$  is chosen, and *Enhanced Greedy* is used to extend the partial cover

into a cover  $S$ . If  $S$  is better than  $S_{best}$ , then  $S_{best} \leftarrow S$ . The algorithm performs well on the reported instances. Another improvement on the greedy algorithm was introduced by Musliu [27]. As with ITEG, an initial cover is produced using the above procedure. A neighborhood of the current solution,  $S$ , is created. A neighbor of this initial cover is a cover that can be found by adding and/or removing columns of  $A$  from the initial cover. A column index  $j$  is put into and saved in a “tabu list” for a specified number of iterations if the  $j$ th column of  $A$  has been added or removed from a cover in one of the recent iterations. An upper bound on the size of the neighborhood is also specified to limit the search time. Once a neighborhood is found, the best neighbor is chosen using a fitness function. For a cover  $D$ , define  $Fitness_D = \text{number of uncovered elements} + |D|$ . The fitness function is among criteria that is used to choose a new cover from the neighbors. The tabu list and upper bound are updated, and the procedure is repeated until stopping criteria is met. According to the computational results reported, the heuristic is competitive with ITEG.

A genetic algorithm developed by Beasley and Chu [8], uses the idea of “survival of the fittest”. The general idea of the genetic algorithm is to produce a population of solutions until a sufficient solution is found. Begin by creating an initial population of solutions, and evaluate the fitness of the solutions. For a solution  $D$ ,  $Fitness_D = |D|$  since  $D$  is a binary vector. Solutions, referred to as parents, are chosen and mated with each other. The children are evaluated, some or all of the population is replaced by

the children. The process of generating children and replacing solutions is repeated as needed. The actual procedure involves a little more work. What follows is a basic description of the algorithm. Once an initial population is generated and  $t$  is initialized to 0, two solutions  $P_1$  and  $P_2$  are mated and a child  $C$  is produced.  $C$  is “mutated”, a random number of chosen elements of  $C$  are switched from 0 to 1 and vice versa, to randomize the search of solutions. If  $C$  is not a solution, a heuristic is used to extend it to a solution. If  $C$  is identical to a solution in the population, the process of producing a child is repeated, else  $t = t + 1$  and a randomly chosen fit solution is replaced by  $C$ . This process is repeated until  $t = M$  for some specified number  $M$ . The fittest solution is chosen as the optimal solution.

Algorithms and heuristics that focus on the polyhedral structure of the SCP have also been developed. In fact, Beasley [7, 9] has conducted further research into solving the SCP using subgradient optimization techniques, problem reduction, and tree search procedures. Balas and Ho [5] also discuss a class of algorithms which incorporate heuristics, cutting planes generated from conditional bounds and subgradient optimization. The heuristics are used to find feasible solutions to the SCP and feasible solutions to the dual program of the linear relaxation of the SCP. The generation of conditional bounds from solutions to the dual program of the linear relaxation of the SCP is discussed by Balas [4]. Balas and Ho ran their algorithms on several randomly generated test instances. According to the reported instances, the algorithm works efficiently in most cases, and provides good approximations when a solution

cannot be found. Further computational experiments are presented by Grossman and Wool [18]. They discuss several approximation algorithms including several versions of greedy algorithms. They provide a basic description of each algorithm and provide results for random generated test instances as well as instances that arise from combinatorial questions. It was reported that the neural network algorithm performed the best overall, and that the greedy algorithms performed competitively in many instances. The best algorithm to use would depend on the test instance.

The methods discussed above are representative of the variety of different approaches that can be taken when faced with solving a SCP. Although many of these traditional methods work well and are viable options to solve many set covering problems, it is not possible to use them directly to solve the SCP that arises from the matroid cogirth problem. This stems from the fact that for all these methods, the entire original system of constraints,  $Ax \geq \mathbb{1}$ , must be known. In fact, at the initialization of the problem of interest, no constraints are known. In the cases of Balas and Ho [5], and Beasley [9], primal and dual heuristics cannot be incorporated into an algorithm. Similarly, finding cutting planes from conditional bounds will not be possible without the dual formulation to the linear relaxation of the SCP. The biggest challenge involving the greedy heuristics and genetic algorithms is the ability to find a good initial solution. Although it might be possible to use genetic algorithms to generate children from an initial population of solutions, generating the initial population would probably be difficult without any weights to decide which columns of  $A$  are



better in the greedy sense. The same can be said for generating a good initial solution for the greedy heuristics discussed at the beginning of this chapter. However, it may be possible to incorporate the underlying concepts into a branch-and-cut algorithm. The greedy heuristics may provide some insight into obtaining feasible solutions to the SCP while only having knowledge of a subset of  $Ax \geq \mathbf{1}$ . It may also be possible to generate other cutting planes without knowing  $Ax \geq \mathbf{1}$  in its entirety. The developed branch-and-cut algorithm will be discussed in Chapter 3. Computational results and observations will be discussed in Chapters 4 and 5.

## Chapter 3

### Methods

#### 3.1 An Algorithm for the Set Covering Problem

As discussed in Chapter 2, there are several exact methods and heuristics to solve the set covering problem (SCP),  $\min \{c^T x \mid Ax \geq \mathbf{1}, x \text{ binary}\}$ . However, these methods require knowledge of the entire system  $Ax \geq \mathbf{1}$ . For the SCP formulated from the cogirth problem, every constraint corresponds to a basis of  $\text{Row}(H)$  and each variable which has a cost of 1 corresponds to a row of  $H$ . Therefore, in order to have the entire system  $Ax \geq \mathbf{1}$ , all the bases of the  $\text{Row}(H)$  must be known. Since this is generally not the case, the previously discussed methods cannot explicitly be used in this case. Instead, I implement a branch-and-cut algorithm, which is a branch-and-bound algorithm that incorporates cutting plane methods, to solve the SCP. I will first briefly discuss the branch-and-bound algorithm and why cutting planes are used.

A branch-and-bound algorithm is designed to solve integer programs to optimality. It begins by considering the linear relaxation of an integer program or mixed-integer program. In this case, consider an integer program  $\min\{c^T x \mid Ax \leq b, x \text{ integer}\}$ . At the root node, before the branching begins, the linear program relaxation  $\min\{c^T x \mid Ax \leq b\}$  is solved. At this point, the solution,  $\bar{x}$ , may be a

mixture of fractional and integral variables. If it is integral and satisfies all explicit and implicit constraints, then the solution is optimal. If not, then the fractional variables are selected and the branching begins. Various branching rules are discussed by Achterberg et al. [2]. At each branch node, a variable is chosen to bound at its upper and lower integral bounds, and a subproblem is created for each bound. The same procedure is used on all the subproblems. Subproblems are pruned (eliminated) if they are infeasible, or if the optimal value of the subproblem is not better than the current best integral solution. Since there are an exponential number of branches, it would be advantageous to find better optimal values early in the procedure. Branching rules can have an effect on the number of branches considered and how quickly optimal solutions are found. Cutting plane methods are used for linear, integer, and mixed integer programs. The idea is to find valid inequalities for the program that will reduce the feasibility region, the solution set of the problem. By reducing the feasibility region, an optimal solution may be found more quickly. A discussion of cutting plane methods can be found in both Nemhauser and Wolsey [28] or Wolsey [35]. However, the general cutting plane methods discussed may not be viable options in this case. In many of the cutting plane generation procedures discussed, the entire constraint set is needed.

Referring back to the SCP that we formulated to solve the matroid cogirth problem, since no bases of  $Row(H)$  are initially known, there are not any constraints. Therefore, we must use a branch-and-cut algorithm to solve the SCP. The branch-

and-cut algorithm is guaranteed to solve the SCP to optimality. A more general discussion of the branch-and-cut algorithm can be found in Wolsey [35]. Figure 3.1 shows a flow chart of the basic algorithm. The implementation of the branch-and-cut algorithm is discussed below. The following subsections describe the intricacies of the algorithm. For the flow chart,  $\min \{c^T x \mid x \in X\}$  is the initial formulation with  $X$  the set of feasible solutions,  $P$  is the original formulation of the problem,  $z$  is the optimal value for the original formulation,  $x^*$  is the optimal solution and  $\bar{z}$  is the bound used to prune solutions.  $P^i$  and  $X^i$  correspond to the formulation and solution set for the  $i$ th subproblem picked from the Node list.  $P^{i,k}$  is the  $k$ th formulation for the  $i$ th subproblem after a cutting plane has been added to it,  $x^{i,k}$  is the solution to the formulation and  $\underline{z}^{i,k}$  is the optimal value.  $P_t^i$  and  $X_t^i$  are subproblems created after a fractional variable has been chosen.

Since no basis is initially known, the algorithm begins at the root node with no constraints; the constraint set, which will be referred to as  $Q$ , is empty. The initial solution to the relaxation, the zero vector, is not a feasible solution to the SCP. At this point a basis of  $Row(H)$  is found and a constraint is added to  $Q$ . A new solution,  $x$ , to the relaxation is found. The values of the individual variables are used as weights for the rows of  $H$  since each variable corresponds to a row. In other words, rows are compiled together one by one by minimum weight until a basis is found. A constraint corresponding to a minimum weight basis  $B$  is added to  $Q$  if  $x$  violates the constraint. That is, if the inner product between the solution  $x$  and the rows corresponding to

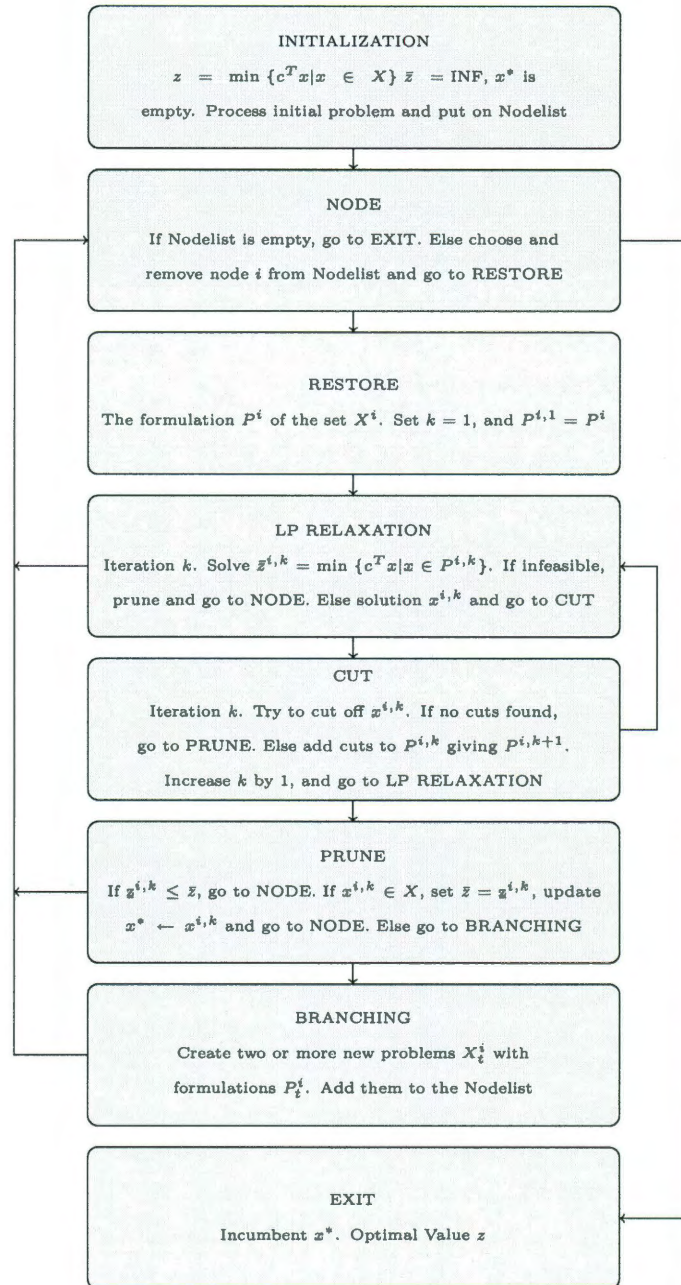


Figure 3.1 : Flow chart of basic Branch-and-Cut Algorithm.

the newly found basis  $B$  is less than 1, the constraint is added. The constraint set  $Q$  is a subset of the system  $Ax \geq \mathbf{1}$ . Because  $x$  is used as weights to find a new basis, it is guaranteed that a newly added constraint was not previously in  $Q$ . If it was, then  $x$  would have already satisfied the corresponding constraint. The idea behind adding constraints this way is that not all the constraints of  $Ax \geq \mathbf{1}$  are needed. Only significant bases are used to add constraints to the system. It would not be advantageous to find every basis of  $H^T$  since there may be a large number of them in general. With this in mind, it is often not the case that the optimal solution to the SCP will be found at the root node since a relaxation of a subset of constraints of the SCP is being solved in the algorithm and the polyhedron for the SCP may not be totally unimodular. Therefore, after adding as many constraints as possible at the root node, the actual branching part of the algorithm is begun. Other than branching on fractional elements of  $x$ , more cutting planes are added at each node of the branch tree. Since the system  $Ax \geq \mathbf{1}$  is not known in its entirety, constraints from the system are added the same way they were at the root node. Other cutting plane methods are also incorporated to find other valid inequalities. The rest of this section will concentrate on cutting plane generation subroutines that were incorporated in the algorithm, the branching rule that was used, and a heuristic that was used to find feasible solutions to the SCP.

### 3.1.1 Generating Cutting Planes

As discussed above, constraints from  $Ax \geq \mathbf{1}$  are added iteratively by finding bases of  $Row(H)$ . The algorithm attempts to add these constraints at the root node and at each branch node. In order to find bases, a greedy algorithm is used. Once a solution,  $x = (x_1, \dots, x_n)^T$ , is found, the elements are sorted. The sorted solution is  $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \dots \leq x_{\sigma(n)}$  where  $\sigma$  is a permutation of the indices. Using the weights, rows of  $H$  are added to a matrix, say  $H_s$ , as long as they increase the rank of  $H_s$ . The rows are added in order of minimum weight until the rank of  $H_s$  is equal to the rank of  $H$ . Once a basis is found, the corresponding constraint is added to constraint set  $Q$  if it is violated by the current solution  $x$ . An initial set of bases is found by first finding a feasible solution to the SCP. The initial set of bases is obtained in such a way that they are as disjoint as possible. The method used to find a feasible solution is discussed in Subsection 3.1.3. Once a feasible solution,  $x$ , to the SCP is found, a greedy algorithm is used to find a set of bases. For each  $j$  such that  $x_j = 1$ , a basis of  $Row(H)$  containing row  $j$  is found. The weight of each row of  $H$  is initially set to zero. Every time a row is added to a basis, its weight is increased by one. Rows with minimum weight are added to a basis first. Bases were found this way to make them as disjoint as possible. After this initial set of bases that are as disjoint as possible, bases are found based on the the solution  $x$  to the linear relaxation. All the constraints obtained from bases are global cuts. The algorithm also attempts to add valid inequalities generated from these basis constraints.

Balas and Ng [6] consider the set covering polytope,  $P_I(A) := \text{conv}\{x \in \mathbb{R}^n \mid Ax \geq \mathbf{1}, x \text{ binary}\}$ . They discuss the following class of inequalities. For a subset of rows  $S$  of the matrix  $A$  from  $Ax \geq \mathbf{1}$ , the inequality  $\alpha^S x \geq 2$  associated with  $S$  is defined by

$$\alpha_j^S = \begin{cases} 0 & \text{if } a_{ij} = 0 \text{ for all } i \in S, \\ 2 & \text{if } a_{ij} = 1 \text{ for all } i \in S, \\ 1 & \text{otherwise.} \end{cases}$$

According to Balas and Ng [6], this class of inequalities, which will be referred to as  $CP$  and is valid for  $P_I(A)$ , can be generated using the following procedure D:

- (i) add the inequalities  $a^i x \geq 1, i \in S$
- (ii) divide the resulting inequality by  $|S| - \epsilon, 0.5 < \epsilon < 1$
- (iii) round up all coefficients to the nearest integer

It would be advantageous to incorporate inequalities from this class. Fortunately, Balas and Ng [6] proved the following theorem. Let  $\bar{x}$  be a fractional solution to the  $Ax \geq \mathbf{1}, 0 \leq x \leq 1$ ,  $R$  be the row indices,  $N$  be the column indices,  $I := \{j \in N \mid \bar{x}_j = 1\}$ , and  $R(I) := \{i \in R \mid a_{ij} = 0, \forall j \in I\}$ .

**Theorem 3.1** *Let  $a^S x \geq 2$  be an inequality in the class  $CP$  that cuts off  $\bar{x}$ . Then  $\alpha_j^S = 0$  for all  $j \in I$ ; i.e.,  $S \subseteq R(I)$*

The theorem states that the search for the set  $S$  of rows associated with the  $\alpha^S x \geq 2$  that cuts off  $\bar{x}$  can be restricted to  $R(I)$ . Procedure D is incorporated into the branch-and-cut algorithm at each branch node. After as many inequalities from  $Ax \geq \mathbf{1}$  have been added to  $Q$ , the theorem proved by Balas and Ng [6] is used to try to



find constraints using procedure D. Since only a subset of the original constraints is known, inequalities from the class  $CP$  may not always be found using this procedure. Further discussion of these cuts and their use in the branch-and-cut algorithm is discussed in Chapter 5.

Beasley et al. [9] also discuss feasible solution exclusion constraints for the SCP. Suppose  $T_c$  is a set of column indices that correspond to the best feasible solution for the SCP. It is assumed without loss of generality that  $T_c - j$  is not a feasible solution for all  $j \in T_c$ . Then according to Balas et al. [9], the following two constraints can be added to the program.

$$\sum_{j \in T_c} x_j \leq |T_c| - 1$$

$$\sum_{j \notin T_c} x_j \geq 1$$

The idea is that if a better solution exists, it can be obtained by replacing at least one column in the current solution. These constraints are incorporated into the branch-and-cut algorithm. These constraints are only considered after branching has begun. After a feasible solution is found using the heuristic discussed below, a check is performed to see if it is better than the current best feasible solution. If so, then two feasible solution exclusion constraints are added. This procedure is done every time the best feasible solution is replaced in a branch node. As I mentioned earlier, Achterberg [2] discusses many possible branching techniques. The next section discusses which rule was used in the proposed branch-and-cut algorithm.

### 3.1.2 Branching

Various branching rules can be incorporated into a branch-and-cut algorithm. The branching rule used for the proposed branch-and-cut algorithm is that the most infeasible variable is chosen to branch on [2]. Since  $0 \leq x \leq 1$ , the closer a variable is to 0.5, the more infeasible it is. The branch-and-cut algorithm implements depth-first search branching, i.e. once a fractional variable, say  $x_k$  is chosen,  $x_k$  is set to zero and the resulting subproblem is explored before  $x_k$  is set to one. The depth-first branching is incorporated in a recursive manor. In order to enhance the branch-and-cut algorithm, a method to find better feasible solutions to the SCP is also included. The method is discussed in the following subsection.

### 3.1.3 Finding Feasible Solutions

Recall that a feasible solution to the SCP is actually a cocircuit of the matroid  $M = (S, \mathcal{I})$  where the ground set  $S$  corresponds to the indices of the rows of  $H$  and  $\mathcal{I}$  is the collection subsets corresponding to linearly independent sets of rows of  $H$ . Therefore, it would be beneficial to find feasible solutions as often as possible without enumerating all possible solutions. In order to find feasible solutions, a greedy algorithm is implemented. The algorithm is similar to that used to find bases. As before, the solution,  $x$ , to the linear program is used as weights. However, in this case, rows with max weight are removed from the matrix  $H$  one at a time until the rank of the resulting matrix is reduced by one. The rows removed represent a

feasible solution to the SCP. The intuition behind removing rows in this manner is that rows with larger weights are more significant to the current relaxed subproblem and may be more likely to be contained in a cocircuit of  $M[H]$ . The only way to ensure that a feasible solution to the SCP found this way represents the smallest cocircuit is to enumerate all possible feasible solutions. However, it is not practical for this method to be used by itself. Instead, this method is used to find feasible solutions of smaller cardinality if possible. In doing so, better upper bounds for the branch-and-cut algorithm can be found and more branch nodes can be pruned earlier in the branch-and-cut algorithm. This method is applied after a new solution  $x$  to the relaxation is found. The next section describes a procedure to arrange a matrix into BBDF. The BBDF is incorporated into a modification of the branch-and-cut algorithm. Computational results of the modified algorithm are presented in Chapter 4.

### 3.2 Rearranging Matrices into Bordered Block Diagonal Form

Recall that part of the method described by Cho et al. [13] involved using a matrix that had been rearranged into bordered block diagonal form (BBDF), Figure 2.1. In particular, Cho et al. [13] were considering sparse matrices when discussing the BBDF. The idea is that if a matrix  $T$  can be arranged into BBDF, then the cocircuit of  $M[T]$  can be found by considering the blocks of the rearranged matrix  $BT$  along

with elements of the border, e.g the submatrices  $[A_1, P_1], [A_2, P_2], \dots, [A_2, P_2]$ . To put the matrix in this form, a set of border rows must be found. To find these border rows we use a method that is discussed by Cho [14]. Create a graph  $\hat{G} = (\hat{V}, \hat{E})$  such that the members of  $\hat{V}$  represent rows of the matrix  $T$  and an edge  $(u, v) \in \hat{E}$  if  $T_{uk}$  and  $T_{vk}$  are both nonzero for some column  $k$  of the matrix  $T$ . That is, an edge exists if rows  $u$  and  $v$  share a common column  $k$ . Once  $\hat{G}$  is created, a minimum separating set of vertices  $S_v$  of  $\hat{G}$  is found. A separating set of vertices is a set  $S \subset \hat{V}$  whose removal disconnects the remaining subgraph. *Menger's Theorem*, which follows, is significant to finding the smallest separating set.

**Theorem 3.2** *Let  $G = (V, E)$  be a graph and  $A, B \subseteq V$ . Then the minimum number of vertices separating  $A$  from  $B$  in  $G$  is equal to the maximum number of disjoint  $A - B$  paths in  $G$ .*

In order to find the smallest separating set, we find the maximum number of  $u - v$  paths for all  $u, v \in \hat{G}$  with  $u \neq v$ . We can do this by solving a series of max-flow min-cut problems from another graph obtained from  $\hat{G}$ . The details can be found in [14]. After the separating set is found the blocks of  $BT$  can be found. Let  $\bar{G} = (V, E)$  be a bipartite graph where members of  $V$  represent rows and columns of  $T$ . Let  $(u, k) \in E$  if  $T_{uk}$  is nonzero for row  $u$  and column  $k$ . Remove the separating set  $S_v$  from  $\bar{G}$  and any edges adjacent to it. The resulting connected components represent the blocks of the matrix  $BT$ . In the following chapters, the branch-and-cut algorithm is run on several test instances. The computational results are presented, and a discussion of

possible avenues for further research is discussed.

## Chapter 4

### Computational Results

In this section, some computational results are run to test the branch-and-cut algorithm, and compare its performance to that of Kianfar et al. [23], one of the methods discussed in Chapter 2. The branch-and-cut algorithm performed on the original matrix is called Algorithm 1. The branch-and-cut algorithm that incorporates the BBDF is called Algorithm 2, and the 0-1 MIP approach of Kianfar et al. [23] is called Algorithm 3. Since Kianfar et al. [23] reported better computation times than Cho et al. [13], their formulation was used as a comparison. The algorithms are compared using computational time.

The results are reported in Table 4.1. Test instance 1 is the same reported by Cho et al. [13]. Test instances 2-6 are those reported by Kianfar et al. [23]. Table 4.1 shows the size of the matrix ( $n \times p$ ), the number of rows in the border,  $|P|$ , the DoR,  $d^*$ , and the computational times for the three algorithms for each test instance. For the test instances in which “> 36000 secs” is seen in the computational times, the algorithms were unable to solve the problem of interest to optimality. The algorithms were stopped manually. Algorithm 2 consists of running the branch-and-cut algorithm on submatrices of the original matrices. In order to find these submatrices, the BBDF of the matrix was computed. In order to give more insight into Algorithms 1 and

2, Tables 4.2 and 4.3 show more detailed computational times for the algorithms. All three algorithms were tested on the same machine, a Dell Precision T3500 Tower Workstation with an *Intel*<sup>®</sup> *Xeon*<sup>®</sup> Dual Core W3505 2.53GHz processor. The algorithms were coded in C++, and Gurobi [19] was used to solve all the linear, and mixed integer program formulations created during the experiments.

Overall, Algorithm 1 performed the poorest compared to the other two algorithms. According to Table 4.2, the best feasible solutions are found within seconds. For test instances 1, 2, and 5, the best feasible solution was an optimal solution. Although the best feasible solution found was not optimal for test instances 3, 4, and 6, the solutions found provided a bound for the optimal solution. Algorithm 2 performed better than Algorithm 3 in test instances 1,2 and 4. Algorithm 3 performed better on the other three instances. In particular, Algorithm 3 appears to perform better when  $d^* \leq 4$  and Algorithm 2 appears to perform better when  $d^* > 4$ . This may indicate that Algorithm 3 is better to use when  $d^*$  is small, and Algorithm 2 may be better to use when  $d^*$  is large and the given matrix has the suggested BBDF structure. Unfortunately, a range for  $d^*$  must be known beforehand in order to decide which algorithm to use. This is highly unlikely. According to Table 4.3, Algorithm 2 found optimal solutions for most of the test instances within a few seconds. In test instances 3-6, the algorithm took considerably longer to finish. This may be due to the number of blocks that have to be considered by the algorithm. In all, Algorithms 2 and 3 are competitive. The key components to the efficiency of Algorithm 2 appear

to be the time it takes to find the border for the BBDF, and the size and number of blocks that need to be solved.

Although the methods discussed in this thesis provide good upper bounds for the matroid cogirth problem, when considering a minimization problem, a lower bound on the optimal solution is more useful than an upper bound. However, a good lower bound is often harder to obtain. Similarly, when considering how many sensors failures are needed to lose the integrity of a sensor network, a lower bound is also more useful. With this in mind, I note a relationship between the spark of a matrix and the girth of a vector matroid. Donoho and Elad [16] define the spark of a matrix as the smallest number of linearly dependent columns of a matrix. Consider a matrix  $F$  and  $M[F]$  with the indices of the columns as the ground set. The circuits of  $M[F]$  are all the sets of minimally dependent columns. Therefore, the girth, cardinality of the smallest circuit, of  $M[F]$  is equal to the spark of  $F$ . Also, the cogirth of  $M[F]$  is equal to the spark of the matrix representation of  $M^*[F]$ , the dual matroid of  $M[F]$ . Donoho and Elad [16] provide a bound on the spark of the matrix. So, one could use this lower bound as a lower bound on the cogirth of a vector matroid. However, the hypergraph developed to describe the cogirth problem-set covering problem relationship can also be used to find a lower bound for cogirth problem. Given the hypergraph describing the vector matroid, recall that a transversal of the hypergraph is a cocircuit of the vector matroid. Given a graph (hypergraph), a *matching* is a set of pairwise non-adjacent edges (hyperedges). The cardinality of a matching, is a



lower bound on the cardinality of the smallest transversal. Therefore, a maximum matching provides the highest lower bound for the transversal number. We note that the optimal value of the linear relaxation found during the branch-and-cut procedure also provides a lower bound. Table 4.4 shows lower bounds using the lower bound described by Donoho and Elad and a lower bound provided by the linear relaxation of the SCP for the matroids obtained from the test instances. The lower bounds obtained from the linear relaxations were obtained using Algorithm 1, and were computed even when Algorithm was not able to solve the SCP to optimality. Note, that in each case, the LP relaxation lower bound is better than the one provided using the method described by Donoho and Elad [16].

I also considered using the formulation presented by Kianfar et al. [23] as a heuristic. Instead of solving the 0-1 MIP as a MIP, for each  $j \in \{1, \dots, p\}$ , set  $z_j = 1$  and relax the binary constraints on  $q_i$  for all  $i \in \{1, \dots, n\}$ ; i.e.,  $0 \leq q_i \leq 1$  for all  $i \in \{1, \dots, n\}$ . Then, solve the resulting sequence of problems for each  $z_j$ . Let  $\{z_{\sigma(1)}, \dots, z_{\sigma(k)}\}$  be the  $k$  variables from  $\{z_1, \dots, z_p\}$  that provide the best  $k$  optimal values when each of the variables was set to 1. Note  $k$  is specified by the user. Since  $\sum_{j=1}^p z_j = 1$  and  $z_j$  is binary for each  $j \in \{1, \dots, p\}$ , only one variable is set to 1 in each instant. Given these  $k$  variables,  $k$  0-1 MIP are solved with the original restriction that  $q_i$  is binary for all  $i \in \{1, \dots, n\}$  and the new restriction  $z_{\sigma(j)} = 1$  for each  $j \in \{1, \dots, k\}$ . In other words, for each  $j \in \{1, \dots, k\}$ , solve the following problem.

$$\min \sum_{i=1}^n q_i$$

$$\begin{aligned}
\text{s.t. } & -q_i \leq \sum_{i=1}^p h_{ij} x_j \leq q_i, \quad i = 1, \dots, n \\
& -1 + 2z_j \leq x_j \leq 1, \quad j = 1, \dots, p \\
& \sum_{i=1}^p z_i = 1 \\
& z_{\sigma(j)} = 1
\end{aligned}$$

Table 4.5 has results for  $k = 5$  and  $k = 10$ . For test instance 6, I set each  $z_j = 1$  for all  $j \in \{1, \dots, p\}$ , left the  $q_i$  binary for all  $i \in \{1, \dots, n\}$ , and solved each problem as a MIP. In all, a total of 252 problems, one for each column of the matrix, were solved. The best optimal value found was 16, not 15 as reported by Kianfar et al. [23]. This was done by setting  $z_j = 1$  for each  $j \in \{1, \dots, p\}$  and setting  $q_1, \dots, q_n$  as binary variables. The optimal value was given with the variable  $z_{218}$  equal to 1. Therefore, confirmation of the optimal value for test instance 6 is needed. In Chapter 5, I present some concluding remarks and other observations.

Matrix H				Computation Time		
No.	Size ( $n \times p$ )	$ P $	$d^*$	Algorithm 1	Algorithm 2	Algorithm 3
1	$34 \times 12$	2	6	56 secs	$\approx 0$ secs	0.08 secs
2	$66 \times 27$	3	7	> 36000 secs	$\approx 0$ secs	11.59 secs
3	$154 \times 72$	2	4	> 36000 secs	561 secs	24.54 secs
4	$221 \times 55$	1	14	> 36000 secs	798 secs	10560 secs
5	$318 \times 144$	2	4	> 36000 secs	918 secs	54.42 secs
6	$1009 \times 252$	1	15 <sup>#</sup>	> 36000 secs	> 36000 secs	> 36000 secs

Table 4.1 : Computational times for Algorithms 1, 2, and 3. # Reported in [23].

Matrix H			Algorithm 1 Comp. Time		
No.	Size ( $n \times p$ )	$d^*$	BFS	BFS Time	Total Time
1	$34 \times 12$	6	6	3 secs	56 secs
2	$66 \times 27$	7	7	14 secs	> 36000 secs
3	$154 \times 72$	4	5	248 secs	> 36000 secs
4	$221 \times 55$	14	16	$\approx 0$ secs	> 36000 secs
5	$318 \times 144$	4	4	1 secs	> 36000 secs
6	$1009 \times 252$	15 <sup>#</sup>	20	13 secs	> 36000 secs

Table 4.2 : Break down of computational times for Algorithms 1. The time for Best Feasible Solution (BFS) is the time it took to find the best feasible solution. # Reported in [23].

Matrix H					Algorithm 2 Comp. Time			
No.	Size ( $n \times p$ )	$ P $	# blocks	$d^*$	BFS	BBDF Time	BFS Time	Total Time
1	$34 \times 12$	2	6	6	6	$\approx 0$ secs	$\approx 0$ secs	$\approx 0$ secs
2	$66 \times 27$	3	12	7	7	$\approx 0$ secs	$\approx 0$ secs	$\approx 0$ secs
3	$154 \times 72$	2	20	4	4	14 secs	1 secs	561 secs
4	$221 \times 55$	1	12	14	14	$\approx 0$ secs	5 secs	798 secs
5	$318 \times 144$	2	38	4	4	151 secs	1 secs	918 secs
6	$1009 \times 252$	1	43	15 <sup>#</sup>	17	$\approx 0$ secs	224 secs	> 36000 secs

Table 4.3 : Break down of computational times for Algorithms 2. The time for Best Feasible Solution (BFS) is the time the solution was found during the entire algorithm, not in a particular block. Except for test instance 6, the value of the BFS was equal to the optimal value. # Reported in [23].

Matrix H				
No.	Size ( $n \times p$ )	$d^*$	Spark LB	LP Relaxation LB
1	$34 \times 12$	6	2	3
2	$66 \times 27$	7	2	3
3	$154 \times 72$	4	2	3
4	$221 \times 55$	14	2	5
5	$318 \times 144$	4	2	3
6	$1009 \times 252$	15 <sup>#</sup>	2	5

Table 4.4 : Lower Bounds provided by Donoho [16] and the LP Relaxation. # Reported in [23].

Matrix H			$k = 5$		$k = 10$	
No.	Size ( $n \times p$ )	$d^*$	computed $d^*$	Comp. Time	computed $d^*$	Comp. Time
1	$34 \times 12$	6	6	1 sec	6	1 sec
2	$66 \times 27$	7	7	1 sec	7	2 sec
3	$154 \times 72$	4	4	3 sec	4	8 sec
4	$221 \times 55$	14	13	9 sec	13	11 sec
5	$318 \times 144$	4	4	12 sec	4	28 sec
6	$1009 \times 252$	15 <sup>#</sup>	17	298 sec	17	572 sec

Table 4.5 : Computational times for Algorithm 3 as a Heuristic. # Reported in [23].

## Chapter 5

### Conclusion

In this thesis, I proposed a branch-and-cut algorithm to solve a set covering problem. In turn, the solution to the set covering problem provides a solution to the matroid cogirth problem for a vector matroid. Although the algorithm is designed to find a solution for the matroid cogirth problem, it can be adapted to find a solution for the matroid girth problem. One simply has to flip the values of the binary constraint matrix  $A$  for the set covering problem.

The results showed that Algorithm 1 did not perform very efficiently on most of the test instances. Some possible reasons are the following. First, as the size of the test matrices grew, the size of a basis for the matroid, and the number of bases grew as well. The number of significant bases required to solve the problems increased and so did the time needed to find them. Also, it was difficult to find cuts from the class of inequalities developed by Balas and Ng [6] discussed in Chapter 3 in both Algorithms 1 and 2. Therefore, it was harder to reduce the feasibility region using these cutting planes. However, the feasible solution exclusion constraints described by Beasley et al. [9] were useful in both Algorithms 1 and 2. Algorithm 2 performed better than Algorithm 1, but this was due to the number and size of matrices in the BBDF of the test matrices. A promising result is that both algorithms were able

to find optimal solutions quickly even though they took longer to terminate. This was partially due to the number of branch nodes that needed to be pruned in both algorithms. Therefore, the algorithms could be manually terminated after a given time, and used as heuristics. As I mentioned in Chapter 4, the 0-1 MIP formulation presented by Kianfar et al. [23] can be used as a heuristic. In fact, the suggested heuristic appears to be accurate for most of the test instances and more efficient than just solving the 0-1 MIP formulation as it was originally presented.

The main goal of this research was to consider a well-known problem from a different perspective. The problem remains difficult to solve. For Algorithms 2 and 3, the ability to find cuts for the system of constraints faster is the biggest challenge. Even though both algorithms found optimal solutions within seconds, exploring as little branch nodes as possible in the both algorithms is also important. Therefore, studying and finding good branch rules is also important. The ability of Algorithms 1 and 2 to find optimal solutions quickly points to the possibility of using them as heuristics. Using the 0-1 MIP formulation in a heuristic should also be considered. Solving the matroid cogirth problem for vector matroids using mixed integer programming techniques appears to be a promising avenue. In any case, ongoing efforts to solve this problem using any sort of MIP formulation should focus on finding strong valid inequalities for the problem.

## Bibliography

- [1] ILOG Company, CPLEX Documentation, Available: <http://www.decf.berkeley.edu/help/apps/ampl/clpex-doc/>, 2007.
- [2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2004.
- [3] M.J. Bagajewicz and M.C. Sanchez. Design and upgrade of nonredundant and redundant linear sensor networks. *AIChE journal*, 45(9):1927–1938, 1999.
- [4] E. Balas. Cutting planes from conditional bounds: A new approach to set covering. In R. W. Cottle, et al., editor, *Combinatorial Optimization*, volume 12 of *Mathematical Programming Studies*, pages 19–36. Springer Berlin Heidelberg, 1980. 10.1007/BFb0120885.
- [5] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. In R. W. Cottle, et al., editor, *Combinatorial Optimization*, volume 12 of *Mathematical Programming Studies*, pages 37–60. Springer Berlin Heidelberg, 1980. 10.1007/BFb0120886.
- [6] E. Balas and S. M. Ng. On the set covering polytope: I. all the facets with coefficients in 0, 1, 2. *Mathematical Programming*, 43:57–69, 1989. 10.1007/BF01582278.
- [7] J. E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85 – 93, 1987.
- [8] J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392 – 404, 1996.
- [9] J. E. Beasley and K. Jornsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58(2):293 – 300, 1992.
- [10] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.
- [11] E. Boros, K. M. Elbassioni, V. Gurvich, and L. Khachiyan. Algorithms for enumerating circuits in matroids. In *Algorithms and Computation, 14th International Symposium, ISAAC 2003*, volume 2906 of *Lecture Notes in Computer Science*, pages 485–494, Kyoto, Japan, December 2003. Springer.



- [12] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1999.
- [13] J. J. Cho, Y. Chen, and Y. Ding. On the (co)girth of a connected matroid. *Discrete Appl. Math.*, 155(18):2456–2470, 2007.
- [14] J.J. Cho. *On the robustness of clustered sensor networks*. PhD thesis, Texas A&M University, 2007.
- [15] V. Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- [16] D. L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via  $\ell_1$  minimization. *Proceedings of the National Academy of Sciences of the United States of America*, 100(5):2197–2202, March 2003.
- [17] S. Govindaraj. Calculation of sensor redundancy degree for linear sensor systems. Master’s thesis, University of Iowa, 2010.
- [18] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81 – 92, 1997.
- [19] Gurobi Optimization Inc. Gurobi Optimizer Reference Manual Version 4.0.0, 2010.
- [20] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256 – 278, 1974.
- [21] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- [22] L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On the complexity of some enumeration problems for matroids. *SIAM J. Discrete Math.*, 19(4):966–984 (electronic), 2005.
- [23] K. Kianfar, A. Pourhabib, and Y. Ding. An integer programming approach for analyzing the measurement redundancy in structured linear systems. *Automation Science and Engineering, IEEE Transactions on*, 8(2):447 –450, april 2011.
- [24] M. Luong, D. Maquin, C. T. Huynh, and J. Ragot. Observability, redundancy, reliability and integrated design of measurement systems. In *Proc of 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA '94*, pages 8–10, 1994.

- [25] E. Marchiori and A. Steenbeek. An iterated heuristic algorithm for the set covering problem, 1998.
- [26] E. Moreno Centeno. Implicit hitting set problems. Master's thesis, University of California at Berkeley, 2006.
- [27] N. Musliu. Local search algorithm for unicost set covering problem. In Moonis Ali and Richard Dapoigny, editors, *Advances in Applied Artificial Intelligence*, volume 4031 of *Lecture Notes in Computer Science*, pages 302–311. Springer Berlin / Heidelberg, 2006. 10.1007/11779568\_34.
- [28] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.
- [29] J. G. Oxley. *Matroid theory*. Oxford Science Publications. The Clarendon Press Oxford University Press, New York, 1992.
- [30] Y. C. Pati, R. Rezaifar, Y. C. Pati R. Rezaifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.
- [31] W. J. Rugh. *Linear System Theory*. Prentice Hall Information and System Science Series. Prentice-Hall, Inc, New Jersey, second edition edition, 1996.
- [32] J.A. Tropp, A., and C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inform. Theory*, 53:4655–4666, 2007.
- [33] V. Václavek and M. Loucka. Selection of measurements necessary to achieve multicomponent mass balances in chemical plant. *Chemical Engineering Science*, 31(12):1199 – 1205, 1976.
- [34] H. Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57(3):509–533, July 1935.
- [35] L. A. Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.