ROBUST SENSOR SELECTION

STRONG DETECTABILITY

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nathaniel T. Woodford

In Partial Fulfillment of the

Requirements for the Degree

of

Master's of Science

December 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Shreyas Sundaram, Chair

School of Electrical and Computer Engineering

Dr. Stanislaw Żak

School of Electrical and Computer Engineering

Dr. Shaoshuai Mou

School of Aeronautics and Astronautics

Approved by:

Dr. Pedro Irazoqui

Head of the School Graduate Program

Ad Veritas

ACKNOWLEDGMENTS

The author thanks his thesis advisor, Shreyas Sundaram, for the countless hours of guidance provided throughout the completion of this research, his father, Todd Woodford, and his mother, Pam Woodford, for teaching their children to love learning, and his wife, Hannah Woodford, for her encouraging support.

Part of the results of this thesis will be appearing in the 2018 IEEE Conference on Decision and Control. This work was supported by NSF grant CMMI-1635014 and by the Purdue Military Research Initiative.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

ABSTRACT viii

1 INTRODUCTION 1

1.1 Motivating Problems 3

1.1.1 The Multi-Processor System-on-Chip Problem 4

1.1.2 The Gas Diffusion Problem 4

1.2 Notation and Terminology 5

2 BACKGROUND KNOWLEDGE - CONTROL THEORY 6

2.1 The Dynamic System Model 6

2.2 Unknown Input Observer (UIO) 6

2.3 Detectability and Observability 7

2.4 Test for Strong Observability 9

2.5 Danger of Invariant Zeros 9

3 BACKGROUND KNOWLEDGE - ALGORITHMS 11

3.1 Complexity 11

3.2 Approximations 12

4 THE STRONG DETECTABILITY SENSOR SELECTION PROBLEM . . . 13

4.1 Problem Formulation 13

4.2 Complexity of SDSS 14

4.3 Complexity of Satisfying the Matching Condition 18

5 THE STRONG DETECTABILITY SENSOR ATTACK PROBLEM 21

5.1 Problem Formulation 21

5.2 Complexity of SDSA 22

Page

6 STRONG DETECTABILITY SENSOR SELECTION PROBLEM APPROXIMATION 25

6.1 Constant Factor Approximation 26

6.2 Synergies 27

7 SIMULATIONS 29

8 SUMMARY AND FUTURE WORK 32

REFERENCES 33

A CODE 36

VITA 46

LIST OF TABLES

Table Page

7.1 SDSS Approximation Over Problem Set 4: n = 400, m = 200, p = 1200, and 2% density 30

7.2 SDSS Approximation Over Problem Set 5: n = 400, m = 200, p = 2200, and 2% density 31

7.3 SDSS Approximation Over Problem Set 6: n = 400, m = 200, p = 1200, and 5% density 31

ABSTRACT

Woodford, Nathaniel T. M.S., Purdue University, December 2018. Robust Sensor Selection Strong Detectability. Major Professor: Shreyas Sundaram.

An unknown input observer provides perfect asymptotic tracking of the state of a system affected by unknown inputs. Such an observer exists (possibly requiring a delay in estimation) if and only if the system satisfies a property known as strong detectability. In this thesis, we consider the problem of selecting (at design-time) a minimum cost subset of sensors from a given set to make a given system strongly detectable. We show this problem is NP-hard even when the system is stable. Furthermore, we show it is not possible to approximate the minimum cost within a factor of $\log n$ in polynomial-time (unless $P = NP$). However, we prove if a given system (with a selected set of sensors) is already strongly detectable, finding the smallest set of additional sensors to install to obtain a zero-delay observer can be done in polynomial time. Next we consider the problem of attacking a set of deployed sensors to remove the property of strong detectability. We show finding the smallest number of sensors to remove is NP-hard. Lastly through simulations, we analyze two greedy approaches for approximating the strong detectability sensor selection problem.

1. INTRODUCTION

There is an increasing need to design controllers and estimators for large-scale systems in a variety of application domains, including computational biology, system of systems, intelligent traffic systems, communication networks, and power grids [1–4]. The states of such systems can be (partially) measured by sensors deployed at various locations. However, there are many instances where it would be difficult or impractical to measure all the states of the system. This could be due to initial implementation cost or runtime energy cost of the sensors [5]. Therefore, a key challenge is finding a subset of sensors with minimum cost to deploy on the system to achieve certain performance objectives.

The problem of determining the minimal cost selection of sensors has been studied extensively in recent years. Existing approaches can be broadly separated into dynamically switching (or scheduling) between different sensors at runtime (e.g., [6–9]), and choosing sensors at design time (e.g., [10–17]). For instance, [10] considered the problem of selecting the smallest number of sensors to make a system observable, and showed this problem is NP-hard to approximate within a factor of $\log n$. In the context of sensor selection for Kalman filtering, the papers [13, 14] showed selecting a set of sensors (within a budget constraint) to minimize the trace of the steady state mean square estimation error (MSEE) is NP-hard, and furthermore, the minimum MSEE cannot be approximated within any constant factor in polynomial-time (unless $P = NP$). Similarly, [15] sought to minimize the number of sensors to achieve a certain estimation error, and to minimize the estimation error with a given number of sensors. The paper [16] studied minimal actuator placement for structural controllability, and [17] took a geometric approach to optimal sensor design.

In this thesis, we consider the problem of (design-time) sensor selection for linear time-invariant systems that are affected by unknown (and arbitrary) inputs. Such

inputs can be used to represent faults, disturbances, model reduction errors, or malicious attacks [18–20]. For instance, in large-scale critical infrastructure and industrial plants, cyber-attacks can be injected at various points in the system, and the characteristics of those attacks may not be known a priori; such attacks can thus be modeled as unknown inputs [19–21]. In this case, the system operator's task is placing sensors on the system in order to estimate the state despite the attacks injected by the adversary. As another example, consider a diffusive process such as a gas spreading over a given area [22], or temperature dynamics across a Multi-Processor-System-on-Chip [23]. These diffusive dynamics are driven by source injections at various locations, whose characteristics may not be known. In such cases, a limited number of sensors must be carefully deployed to estimate the gas concentrations or temperatures at all points in the space, despite lack of knowledge of the injected quantities.

In order to obtain perfect (asymptotic) estimation of the state of systems driven by unknown inputs (such as those described above), one must construct an *unknown input observer*(UIO) which monitors the output of the system (provided by the deployed sensors) and maintains an estimate of the state (possibly with some delay) [24]. Such observers also find applications in fault-detection and robust estimation [25,26]. For an unknown input observer to exist, the system must be *strongly detectable* (i.e., all invariant zeros of the system must be stable) [27, 28]. As a necessary condition for strong detectability is detectability, and it was shown in [10] that it is NP-hard to determine the minimum set of sensors to make a system detectable, the problem that we consider in this thesis is trivially NP-hard as well. However, the fundamental question that motivates this thesis is the following: is the NP-hardness of the sensor selection problem for strong detectability solely due to the need to obtain detectability? In other words, **do the unknown inputs contribute to the computational complexity of the problem?**

We answer this latter question in the affirmative by showing it is NP-hard to find a minimum cost selection of sensors to make a given system strongly detectable,

even when the system is stable. In particular, by restricting our attention to stable systems, we ensure that all sensor selections cause the system to be detectable, and thereby eliminate the complexity of choosing sensors to satisfy that property. Our NP-hardness proof relies on carefully constructed stable LTI system instances affected by unknown inputs, along with sets of available sensors. Additionally, we show the stronger result that the minimum cost cannot be approximated within a factor of $\log n$. This inapproximability result mirrors the corresponding result for minimal sensor selection for observability provided in [10], but again arises from the need to handle the unknown inputs (as opposed to ensuring detectability as in [10]). However, we show once a set of sensors is selected to make the system strongly detectable, the problem of finding the lowest cost set of additional sensors to obtain a zero-delay unknown input observer can be solved in polynomial time.

After establishing the above complexity results for the sensor selection problem, we turn our attention to the problem of *attacking* a deployed set of sensors in order to *remove* the property of strong detectability from the system. Specifically, we consider a scenario where an attacker can remove a given number of deployed sensors in an attempt to cause the remaining system to not be strongly detectable. We prove it is NP-hard for the attacker to find the minimum number of sensors to remove to achieve this.

Finally, we propose two greedy selection criteria to select at design time a minimal cost subset of sensors such that the system is strongly detectable. We created instances of our problem from known instances of Set Cover from paper [29]. Lastly, we provide simulation results comparing the performance of the two greedy selection criteria.

1.1 Motivating Problems

We present two examples of problems where an unknown input observer is required to provide asymptotic knowledge of the state of the system. The first example

considers modeling the temperature of several processors and the second example explores the use of an unknown input observer to determine the gas concentration at various locations.

1.1.1 The Multi-Processor System-on-Chip Problem

Suppose a computer manufacturer is constructing a multi-processor system on a chip. To avoid the system from overheating, the designer desires to know the individual temperature of each processor on the chip.

When relatively few processors exist, the designer could potentially place a temperature sensor on each processors. However, as the number of states increases (i.e., temperature of each processor), it becomes impractical to place a sensor on every processor. This could be due to budget, time, or other constraints. We will not assume the power consumed by all processors is known or accurate (the power consumed relates to the heat output of each processor [30]). Without knowledge of these inputs, a UIO must be designed to have full asymptotic knowledge of each state.

1.1.2 The Gas Diffusion Problem

US military personal wear Mission-oriented protective posture (MOPP) gear in toxic environments. The bulky gear degrades soldiers' performance and increases the time required to complete the mission [31]. Consider a scenario where a commander desires to estimate the concentration of a toxic gas over a large battlefield. By estimating the concentrations, only the soldiers at risk of exposure would be required to wear the MOPP gear. The other soldiers would be able to continue the mission unencumbered.

In the above scenario, the system's states are the gas concentrations of evenly spaced locations on the battlefield. The system's state dynamics are determined by gas diffusion equations. Although the location of each gas source is known, the

amount of gas produced at each source is unknown. Thus this problem also requires a UIO.

We assume due to budget constraints, the commander does not desire to place a sensor over every area of interest on the battle field. Instead the commander desires to know the minimum number of locations a sensor must be placed for a UIO to provide perfect asymptotic tracking of the concentration of the gas at every location of interest. Since the observer does not have access to the inputs (i.e., gas injections) the chemical sensors must be deployed in such a way that a UIO can be constructed.

1.2 Notation and Terminology

The set of real numbers, complex numbers, and integers are denoted as $\mathbb{R}$, $\mathbb{C}$ and $\mathbb{Z}$ respectively. We denote restrictions of those sets via subscripts (e.g., $\mathbb{R}_{\geq 0}$ denotes all nonnegative real numbers). Matrices are denoted in bold (e.g., $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$). The identity matrix of dimension $r \times r$ is denoted $\mathbf{I}_r$ and the zero matrix is denoted as $\mathbf{0}$ (with subscripts to denote the dimensions, as needed). The notation $\mathbf{A}(i, j)$ indicates the i^{th} row and j^{th} column of the matrix $\mathbf{A}$. We denote the transpose of a matrix $\mathbf{A}$ by $\mathbf{A}'$. The notation diag() indicates a diagonal matrix with the values in the parentheses along the diagonal. A binary indicator vector μ is a vector where each element is either a 1 or a 0. The complement of an indicator vector is denoted μ^c, where each 1 in μ becomes a 0 and vice versa. All vectors are column vectors, unless otherwise noted. The symbol e represents Euler's number.

2. BACKGROUND KNOWLEDGE - CONTROL THEORY

2.1 The Dynamic System Model

In this thesis we will focus our attention on the discrete time linear system modeled as follows:

$$\mathbf{x}[t+1] = \mathbf{A}\mathbf{x}[t] + \mathbf{B}\mathbf{u}[t] \tag{2.1}$$

$$\mathbf{y}[t] = \mathbf{C}\mathbf{x}[t], \tag{2.2}$$

where $t \in \mathbb{Z}_{\geq 0}$ is the discrete-time index, $\mathbf{x}[t] \in \mathbb{R}^n$ is the state vector, $\mathbf{u}[t] \in \mathbb{R}^m$ is the unknown input vector, $\mathbf{y}[t] \in \mathbb{R}^p$ is the output vector, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the system dynamics matrix, $\mathbf{B} \in \mathbb{R}^{n \times m}$ is the input matrix, and $\mathbf{C} \in \mathbb{R}^{p \times n}$ is the output matrix. We assume without loss of generality throughout that $\mathbf{B}$ has full column rank.

As stated in (2.1), the state of the system $\mathbf{x}$ changes at each time step. This change is affected by the system dynamics of the current state and also the inputs into the system. The sensors of the system are represented in the $\mathbf{C}$ matrix. As equation (2.2) states, the output of the sensors is a function of the sensor matrix and the states of the system.

2.2 Unknown Input Observer (UIO)

A UIO is a state estimator that takes as input the output vector $\mathbf{Y}$ over several time steps and outputs an asymptotic tracking of the system states with some delay. In particular, the UIO does not know, or use, the input vector $\mathbf{u}$. For $\alpha \in \mathbb{Z}_{\geq 0}$, we will denote the output of system (2.1), (2.2) over α time-steps by:

$$\mathbf{Y}[t : t+\alpha] \triangleq \begin{bmatrix} \mathbf{y}[t]' & \mathbf{y}[t+1]' & \cdots & \mathbf{y}[t+\alpha]' \end{bmatrix}'.$$

The linear system:

$$\begin{aligned} \mathbf{z}[t+1] &= \mathbf{E}\mathbf{z}[t] + \mathbf{F}\mathbf{Y}[t:t+\alpha] \\ \psi[t] &= \mathbf{z}[t] + \mathbf{G}\mathbf{Y}[t:t+\alpha], \end{aligned} \tag{2.3}$$

where $\mathbf{E}$, $\mathbf{F}$ and $\mathbf{G}$ are appropriate matrices, is said to be an *unknown input observer* (UIO) with delay α for the system (2.1)-(2.2) if $\|\mathbf{x}[t] - \psi[t]\| \to 0$ as $t \to \infty$ [28].

2.3 Detectability and Observability

Four related but fundamentally different terms will appear throughout this thesis, detectability, observability, strong detectability, and strong observability. The definition of these terms is as follows:

A system is detectable if and only if:

$$\text{rank} \begin{bmatrix} (\lambda \mathbf{I} - \mathbf{A}) \\ \mathbf{C} \end{bmatrix} = n \quad \forall\ \lambda_i \in \mathbb{C},\ |\lambda_i| \geq 1. \tag{2.4}$$

An observer on a detectable system identifies an output of $\mathbf{y} = \mathbf{0}$ for all time as a state $\mathbf{x}$ that is heading towards zero.

A system is observable if and only if:

$$\text{rank} \begin{bmatrix} (\lambda \mathbf{I} - \mathbf{A}) \\ \mathbf{C} \end{bmatrix} = n \quad \forall\ \lambda_i \in \mathbb{C}. \tag{2.5}$$

An observer on an observable system identifies an output of $\mathbf{y} = \mathbf{0}$ for all time as a state $\mathbf{x}$ that equals zero.

A complex number z_0 satisfying:

$$\text{rank} \begin{bmatrix} \mathbf{A} - z_0 \mathbf{I_n} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} < n + m$$

is said to be an **invariant zero** of the system.

A system is strongly detectable if and only if:

$$\text{rank} \begin{bmatrix} \mathbf{A} - z_0 \mathbf{I_n} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} = n + m, \quad \forall\ z_0 \in \mathbb{C},\ |z_0| \geq 1. \tag{2.6}$$

A system is strongly observable if and only if:

$$\text{rank} \begin{bmatrix} \mathbf{A} - z_0\mathbf{I_n} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} = n + m, \quad \forall\ z_0 \in \mathbb{C}. \tag{2.7}$$

Ideally an observer design will not require sensors measuring every state of the system. Consequently, the $\mathbf{C}$ matrix will not be full rank. Some states $\mathbf{x}[t]$ will be in the null space of $\mathbf{C}$. The $\mathbf{y}$ vector in equation (2.2) could equal $\mathbf{0}$ for all time when either $\mathbf{x} = \mathbf{0}$ or there exist an $\mathbf{x}$ such that $\mathbf{Cx} = \mathbf{0}$ for all time. For an observer to distinguish between these two conditions the system must satisfy the property of observability. If the observer does not use or have access to the inputs, the system must satisfy the property of strong observability. A strongly observable system does not have any invariant zeros.

A less demanding property is detectability. If a system is detectable, an observer on the system given an output of $\mathbf{0}$ for all time would distinguish if $\mathbf{x} \to \mathbf{0}$ or if there exist an $\mathbf{x}$ such that $\mathbf{Cx} = \mathbf{0}$. The state might not be necessarily $\mathbf{0}$ at any point in time but it is heading towards zero. Similar to its observable counterpart, strong detectability is required for observers that do not know or use inputs. A strongly detectable system does not have any unstable invariant zeros (i.e., $|z_0| \geq 1$).

Theorem 2.3.1 ([27, 28]) *An unknown input observer exists for system* (2.1)-(2.2) *if and only if:*

$$\text{rank} \begin{bmatrix} \mathbf{A} - z_0\mathbf{I_n} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} = n + m \qquad \forall\ |z_0| \geq 1. \tag{2.8}$$

Furthermore, if this condition is satisfied, the delay required for the observer will satisfy $\alpha \leq n - 1$. *A zero-delay observer (i.e., with* $\alpha = 0$*) exists if and only if condition* (2.8) *holds, and in addition:*

$$\text{rank}(\mathbf{CB}) = \text{rank}(\mathbf{B}). \tag{2.9}$$

We will refer to condition (2.8) as the **strong detectability condition**, and to (2.9) as the **matching condition**.

2.4 Test for Strong Observability

For any $L \in \mathbb{Z}_{\geq 1}$, the observability matrix $\mathbf{O_L}$ is defined as:

$$\mathbf{O}_L = \begin{bmatrix} \mathbf{C} \\ \mathbf{O}_{L-1}\mathbf{A} \end{bmatrix}, \text{where } \mathbf{O}_1 = \mathbf{C}. \tag{2.10}$$

For any $L \in \mathbb{Z}_{\geq 1}$The invertability matrix $\mathbf{J}_L$ is defined as:

$$\mathbf{J}_L = \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{O}_{L-1}\mathbf{B} & \mathbf{J}_{L-1} \end{bmatrix}, \text{where } \mathbf{J}_1 = \mathbf{D}. \tag{2.11}$$

The $[\mathbf{O}\ \mathbf{J}]_L$ matrix represents the concatenation of the $\mathbf{O}_L$ and $\mathbf{J}_L$ matrices (i.e., $[\mathbf{O}_L, \mathbf{J}_L]$). For ease of notation, at times the $[\mathbf{O}\ \mathbf{J}]_n$, $\mathbf{O}_n$, $\mathbf{J}_n$ matrices are represented as $[\mathbf{O}\ \mathbf{J}]$, $\mathbf{O}$, and $\mathbf{J}$ respectively. To avoid having to test every value of z_0 to determine if a system is strongly observable, the following theorem is convenient.

Theorem 2.4.1 ([32, 33]) *A system is strongly observable if and only if:*

$$\text{rank}([\mathbf{O}\ \mathbf{J}]_L) - \text{rank}(\mathbf{J}_L) = n \text{ for some } L \leq n \tag{2.12}$$

where n equals the number of states.

2.5 Danger of Invariant Zeros

The following example illustrates the effect of an invariant zero on an observer. Consider the following system matrices:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}.$$

We have:

$$\mathbf{O} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 2 & 1 \\ 2 & -2 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{J} = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 2 & -1 & 0 \end{bmatrix}.$$

The system is observable since the $\mathbf{O}_n$ matrix is full rank. However, the difference between the rank of $[\mathbf{O}\ \mathbf{J}]_n$ and $\mathbf{J}_n$ is less than n and therefore the system is not strongly observable. Suppose the initial state of the system and the following inputs are as follows:

$$\mathbf{x}[0] = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{u}[0] = 2 \quad \text{and} \quad \mathbf{u}[t] = 2\mathbf{u}[t-1].$$

This carefully selected initial condition and sequence of inputs will cause the output of the sensor to always equal zero although the state of the system is increasing exponentially with each time step. The state of the system will progress as follows:

$$\mathbf{x}[0] = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{x}[1] = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \ \ldots, \ \mathbf{x}[t] = \begin{bmatrix} 2^t \\ 2^t \\ 0 \end{bmatrix}.$$

The state of the system at each time step is orthogonal to the $\mathbf{C}$ matrix. Therefore, the sensor output will remain zero as the state exponentially increases, thus giving the observer no information about the increase of the state with each time step.

After studying the dynamics of the system the enemy could identify where the invariant zero exits and exploit this weakness. By injecting a series of carefully selected inputs into the system, the system could be guided into a state that exponentially grows without knowledge of the observer. In order to protect from such attacks, the system must exhibit the property of strong observability or at least strong detectability.

3. BACKGROUND KNOWLEDGE - ALGORITHMS

The following definitions are adapted from *Introduction to Algorithms* by Cormen et al. [34].

3.1 Complexity

Definition 3.1.1 *Optimization Problem: A problem whose objective is to maximize or minimize a value. The problem may be subject to various constraints.*

Definition 3.1.2 *Decision Problem: A decision problem where the solution is either a "yes" or "no" or more formally a '1' or a '0'. A decision problem can be viewed as a mapping from an instance I to the set* $\{1, 0\}$.

An optimization problem can usually be formulated as a related decision problem by selecting a limit for the optimal value. A decision problem can provide an answer to an optimization problem through multiple iterations of a binary search. Similarly, a solution to an optimization problem can determine the answer to a corresponding decision problem by comparing the optimal value to the decision problem. The complexity classes $\mathbf{P}, \mathbf{NP}$, and $\mathbf{NP-complete}$ are defined only for decision problems.

Definition 3.1.3 *Complexity Class* **P***: The set of decision problems such that the number of computations required for an algorithm to solve an instance of a decision problem is* $O(n^k)$ *where* n *is the size of the instance and* $k > 0$ *is a problem dependent constant.*

Definition 3.1.4 *Complexity Class* **NP***: The set of decision problems such that a "yes" answer to an instance of the problem can be verified in polynomial time.*

Definition 3.1.5 *Complexity Class* $\mathbf{NP-complete}$*: A problem* P_1 *is* $\mathbf{NP-complete}$ *if* P_1 *satisfies the following two properties. 1)* $P_1 \in \mathbf{NP}$ *2) Any problem* P_2 *in* $\mathbf{NP}$ *can be reduced to an instance of* P_1 *in polynomial time. If* P_1 *satisfies property 2 and not necessarily property 1, it is considered* $\mathbf{NP-hard}$*.*

There is currently no known polynomial-time algorithm that can solve problems in the class $\mathbf{NP-hard}$ and $\mathbf{NP-complete}$. However, no proof exists that states that they cannot be solved in polynomial time. Generally when a problem is proven to be of the class $\mathbf{NP-complete}$ or $\mathbf{NP-hard}$, instead of working to build a polynomial time exact algorithm for the problem a polynomial time approximation algorithm is formulated instead.

3.2 Approximations

The exact solution to a problem can be found by trying every single possibility until a solution is obtained. However, as the size of the problem increases it becomes impractical to use such brute force methods. To find solutions to these problems, approximation algorithms are generally used.

Definition 3.2.1 *A* $f(n)$*-factor approximation algorithm: A polynomial-time algorithm that guarantees a solution to a optimization problem within* $O(f(n))$ *of the optimal solution where* n *is proportional to the size of an instance of the problem.*

Another class of algorithms are heuristic algorithms. These algorithms may outperform a $f(n)$-factor approximation algorithm in practice. However, there may not be a theoretical bound for the algorithms' performance. In many actual applications, a heuristic algorithm and a $f(n)$-approximation algorithm may be run side by side, and the output of the algorithm that gives the best solution may than be chosen.

4. THE STRONG DETECTABILITY SENSOR SELECTION PROBLEM

4.1 Problem Formulation

Consider again system (2.1), and suppose that there are no sensors deployed on the system (i.e., the output equation (2.2) is not initially given). Instead suppose that we have a set $\mathcal{S} = \{S_1, \ldots, S_p\}$ of available sensors and a cost vector $\mathbf{b} \in \mathbb{R}^p_{\geq 0}$ assigning a nonnegative cost to each sensor. In other words, the i^{th} element of $\mathbf{b}$ denotes the cost of sensor S_i, for each $1 \leq i \leq p$.

Each sensor $S_i \in \mathcal{S}$ provides a scalar measurement of the state given by:

$$\mathbf{y}_i[t] = \mathbf{C}_i \mathbf{x}[t], \tag{4.1}$$

for a row vector $\mathbf{C}_i$. Let $\mathbf{C} = \begin{bmatrix} \mathbf{C}_1' & \mathbf{C}_2' & \cdots & \mathbf{C}_p' \end{bmatrix}'$. Given an indicator vector $\mu \in \{0,1\}^p$, we denote $\mathbf{C}(\mu)$ to be the submatrix of $\mathbf{C}$ consisting of the rows corresponding to the sensors indicated by μ.

We consider the following problem.

Problem 1 (Strong Detectability Sensor Selection (SDSS)) *Suppose we are given the system matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, the input matrix $\mathbf{B} \in \mathbb{R}^{n \times m}$, a set of p available sensors $\mathcal{S}$ whose measurements are given by the rows of matrix $\mathbf{C} \in \mathbb{R}^{p \times n}$, and a cost vector $\mathbf{b} \in \mathbb{R}^p_{\geq 0}$. The Strong Detectability Sensor Selection Problem (SDSS) is to solve:*

$$\min_{\mu \in \{0,1\}^p} \mathbf{b}'\mu$$

$$s.t. \quad \operatorname{rank} \begin{bmatrix} \mathbf{A} - z_0 \mathbf{I_n} & \mathbf{B} \\ \mathbf{C}(\mu) & \mathbf{0} \end{bmatrix} = n + m, \quad \forall z_0 \in \mathbb{C},\ |z_0| \geq 1.$$

4.2 Complexity of SDSS

We start by showing the SDSS problem is NP-hard by providing a reduction from Set Cover, stated below.

Problem 2 (Set Cover) *Consider a tuple $(\mathcal{U}, \mathcal{H}, k)$, where $\mathcal{U}$ is a finite set of r elements, $\mathcal{H}$ is a collection of sets $\{H_1, H_2, \ldots, H_q\}$ such that $H_i \subset \mathcal{U}$ for all $i \in \{1, 2, \ldots, q\}$, and k is a nonnegative integer.*

Question: *Do at most k sets from $\mathcal{H}$ exist whose union is equal to $\mathcal{U}$?*

Set Cover is NP-hard [35]. We will now provide a reduction from Set Cover to SDSS, prove certain useful properties of the created instance of SDSS, and consequently prove that SDSS is NP-hard.

Polynomial-Time Reduction from Set Cover to SDSS

Given an instance of Set Cover (with r elements in the set $\mathcal{U}$, and a collection $\mathcal{H}$ containing q subsets of $\mathcal{U}$), we will create an instance of SDSS as follows. Define the matrices $\mathbf{A}$, $\mathbf{A_1}$ and $\mathbf{B}$ as:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{0}_{r\times r} & \mathbf{A_1} \\ \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} \end{bmatrix}, \quad \mathbf{A_1} = \operatorname{diag}(1, 2, \ldots, r), \\ \mathbf{B} &= \begin{bmatrix} \mathbf{I}_r \\ -\mathbf{I}_r \end{bmatrix}. \end{aligned} \tag{4.2}$$

Next, we define a $q \times r$ matrix $\mathbf{S}$ to encode the set $\mathcal{H}$. Each column of $\mathbf{S}$ corresponds to an element in $\mathcal{U}$, and each row encodes one of the sets H_i in the collection $\mathcal{H}$. Specifically, element (i, j) of $\mathbf{S}$ is equal to 1 if set $H_i \in \mathcal{H}$ contains the element $j \in \mathcal{U}$, and zero otherwise.

Now we define the set of sensors for SDSS. Specifically, we create $p = r + q$ sensors to choose from. Each sensor i's measurement matrix $\mathbf{C}_i$ consists of a single row; the collection of the measurement matrices for all sensors is given by:

$$\mathbf{C} = \begin{bmatrix} \mathbf{I}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}. \tag{4.3}$$

Each sensor's cost is defined by an element of the column vector $\mathbf{b}$. In this instance, we set the first r elements of $\mathbf{b}$ to '0', and the remaining elements to '1'.

Properties of the Created Instance

Consider the set of available sensors in the created instance, given by the rows of the matrix $\mathbf{C}$ in (4.3). Since the first r sensors all have zero cost (as specified in $\mathbf{b}$), they can always be included in any sensor selection without increasing the total cost. Thus, the indicator vector μ for the sensor selection will be assumed to have a '1' in each of its first r elements. The matrix $\mathbf{C}(\mu)$ containing the rows of all sensors selected by μ is given by:

$$\mathbf{C}(\mu) = \begin{bmatrix} \mathbf{I}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{S}(\mu) \end{bmatrix}. \tag{4.4}$$

$\mathbf{S}(\mu)$ is the matrix containing those rows of $\mathbf{S}$ that are included in the sensor selection μ.

Lemma 1 *Consider a sensor selection μ and the corresponding tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$, where $\mathbf{A}$ and $\mathbf{B}$ are given by* (4.2) *and $\mathbf{C}(\mu)$ is given by* (4.4)*. All invariant zeros of this tuple (if they exist) are unstable.*

Proof Suppose z_0 is an invariant zero of the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$, i.e., there exists a nonzero vector $\begin{bmatrix} \mathbf{X}_0' & \mathbf{U}_0' \end{bmatrix}'$ such that:

$$\begin{bmatrix} \mathbf{A} - z_0\mathbf{I}_{2r} & \mathbf{B} \\ \mathbf{C}(\mu) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{X_0} \\ \mathbf{U_0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{4.5}$$

By substituting the expressions for $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}(\mu)$ from (4.2) and (4.4) into equation (4.5) and partitioning $\mathbf{X}_0$ as $\begin{bmatrix} \mathbf{X_1}' & \mathbf{X_2}' \end{bmatrix}'$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ each have r elements, the expression (4.5) becomes:

$$\left[\begin{array}{cc|c} -z_0\mathbf{I}_r & \mathbf{A_1} & \mathbf{I}_r \\ \mathbf{0} & -z_0\mathbf{I}_r & -\mathbf{I}_r \\ \hline \mathbf{I}_r & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}(\mu) & \mathbf{0} \end{array}\right] \begin{bmatrix} \mathbf{X_1} \\ \mathbf{X_2} \\ \mathbf{U_0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{4.6}$$

The above expression shows that $\mathbf{X_1} = 0$ and $\mathbf{U_0} = -z_0\mathbf{X_2}$. The first block row in (4.6) then leads to:

$$\mathbf{A_1}\mathbf{X_2} = z_0\mathbf{X_2}. \tag{4.7}$$

This implies that either $\mathbf{X}_2$ is the zero vector, or that z_0 and $\mathbf{X_2}$ are an eigenvalue and corresponding eigenvector of $\mathbf{A_1}$. The former case is impossible, since then $\mathbf{U}_0$ would also be zero in (4.6), contradicting the fact that all three of $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{U}_0$ cannot be zero. Thus, z_0 and $\mathbf{X}_2$ must be an eigenvalue and eigenvector of $\mathbf{A}_1$ respectively. Since all eigenvalues of $\mathbf{A}_1$ in (4.2) are unstable, the claim follows. ■

Lemma 2 *The tuple* $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$ *has no invariant zeros if and only if all columns of* $\mathbf{S}(\mu)$ *are nonzero.*

Proof Suppose all columns of $\mathbf{S}(\mu)$ are nonzero, and assume by way of contradiction that there exists an invariant zero z_0. Thus, there exists a nonzero vector $\begin{bmatrix} \mathbf{X}_1' & \mathbf{X}_2' & \mathbf{U}_0' \end{bmatrix}'$ satisfying (4.6). Furthermore, from the proof of Lemma 1, we know that $\mathbf{X}_2$ is an eigenvector of $\mathbf{A}_1$. Since each eigenvector of $\mathbf{A}_1$ has exactly one nonzero element (by design, from (4.2)), the quantity $\mathbf{S}(\mu)\mathbf{X}_2$ will be a scaled version of a column of $\mathbf{S}(\mu)$. However, if all columns of $\mathbf{S}(\mu)$ are nonzero, then this contradicts the last row of (4.5) and thus there cannot be any invariant zeros of the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$.

Conversely, suppose there is a column of $\mathbf{S}(\mu)$ that is zero. Select $\mathbf{X}_2$ to be the indicator vector with a '1' in the position corresponding to that column, and zeros

everywhere else. Set z_0 to be the eigenvalue of $\mathbf{A}_1$ corresponding to the eigenvector $\mathbf{X}_2$, $\mathbf{X}_1 = 0$, and $\mathbf{U}_0 = -z_0\mathbf{X}_2$. We see that this choice of z_0, $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{U}_0$ satisfy (4.5). Thus, the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$ will have an unstable invariant zero. ■

NP-hardness of SDSS

Using the reduction from Set Cover given by the system (4.2), the set of sensors (4.3) and cost vector $\mathbf{b}$, along with the properties of such instances given above, we obtain the following result.

Theorem 4.2.1 *Given an instance of Set Cover and the associated instance of SDSS (given by* (4.2)*,* (4.3) *and the cost vector* $\mathbf{b}$*), there exists a sensor selection of cost k that makes the system strongly detectable if and only if a set cover of size k or less exists. Thus, SDSS is NP-hard.*

Proof Suppose there exists a set cover of size k or less. Let μ be the sensor selection vector that selects the first r sensors from (4.3) and the k sensors from the bottom q rows of $\mathbf{b}$ to correspond to the elements in the set cover instance. By the definition of the costs in $\mathbf{b}$, this selection has total cost k. Since each row of $\mathbf{S}$ encodes a different subset in the given instance of Set Cover, we see that $\mathbf{S}(\mu)$ will have no empty columns. From Lemma 2, if there are no empty columns in $\mathbf{S}(\mu)$ the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$ will have no invariant zeros, and thus will be strongly detectable.

Now suppose that there is no set cover of size k. Then, for any sensor selection μ of cost k or less, there will be at least one column of $\mathbf{S}(\mu)$ that is zero. From Lemmas 1 and 2, we see that the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu))$ will have an unstable invariant zero, and thus will not be strongly detectable.

Thus, we see that given any instance of Set Cover, we can create an instance of SDSS in polynomial-time, and solve the Set Cover instance by solving the sensor selection instance. Since Set Cover is NP-hard, SDSS is as well. ■

4.3 Complexity of Satisfying the Matching Condition

As the SDSS problem is NP-hard (as shown in Theorem 4.2.1), it is also NP-hard to find a minimum cost selection of sensors in order to construct a UIO (by Theorem 2.3.1). However, suppose that we consider a system that already has a set of sensors deployed which make the system strongly detectable, but that the matching condition (2.9) is not satisfied (so that a zero-delay UIO cannot be created). Suppose that we wish to deploy additional sensors (from a given set) of lowest cost in order to obtain a zero-delay UIO. This requires that the total set of deployed sensors satisfy the matching condition (2.9). In this section, we show that when each sensor provides a scalar measurement of the state (i.e., $\mathbf{C}_i$ in (4.1) is a row vector), a minimum cost set of sensors to satisfy the matching condition can be found in polynomial time. We will use the following result.

Lemma 3 ([34]) *Consider a set $\mathcal{V} = \{v_1, v_2, \ldots, v_p\}$ consisting of p vectors, and a weight $\mathbf{w}_i \in \mathbb{R}_{\geq 0}$ for each vector $v_i \in \mathcal{V}$. The problem of finding the lowest cost maximal linearly independent subset[1] of vectors can be solved in polynomial time via a greedy algorithm.*

We will start by considering the general problem of selecting a subset of sensors of lowest cost in order to satisfy only the matching condition (i.e., without considering the strong detectability condition).

Theorem 4.3.1 *Consider a set of sensors $\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$, where each sensor provides a scalar measurement of the form* (4.1)*. Let the vector $\mathbf{b} \in \mathbb{R}^p_{\geq 0}$ contain the cost of each sensor. Let $\mathbf{C}$ be the matrix consisting of all of the individual sensor matrices. Then, the sensor selection vector $\mu \in \{0,1\}^p$ of lowest cost satisfying the matching condition* $\operatorname{rank}(\mathbf{C}(\mu)\mathbf{B}) = \operatorname{rank}(\mathbf{B})$ *(if such a selection exists) can be found in polynomial time via a greedy algorithm.*

[1] A maximal linearly independent subset of vectors is a linearly independent subset of $\mathcal{V}$ such that no additional vectors from $\mathcal{V}$ can be added to the subset without violating linear independence.

Proof Define the matrix $\mathbf{W} = \mathbf{CB}$, where the i^{th} row of $\mathbf{W}$ is given by $\mathbf{C}_i\mathbf{B}$. Thus, define the cost of the i^{th} row of $\mathbf{W}$ to be $\mathbf{b}_i$, i.e., the cost of the corresponding sensor S_i.

For any sensor selection vector μ, define $\mathbf{W}(\mu) = \mathbf{C}(\mu)\mathbf{B}$, which implies $\text{rank}(\mathbf{W}(\mu)) = \text{rank}(\mathbf{C}(\mu)\mathbf{B})$. Thus, finding a set of rows of $\mathbf{C}$ of minimum cost such that $\text{rank}(\mathbf{C}(\mu)\mathbf{B}) = \text{rank}(\mathbf{B})$ (if it exists) is equivalent to finding a maximal linearly independent set of rows of $\mathbf{W}$ of lowest cost. By Lemma 3 this can be done in polynomial-time via a greedy algorithm. Thus, the lowest cost set of sensors to satisfy the matching condition can be found in polynomial time. ■

Algorithm 1 is an example of a greedy algorithm that takes matrices $\mathbf{B}$ and $\mathbf{C}$, along with a cost for each row of $\mathbf{C}$, and returns a lowest cost sensor selection μ satisfying $\text{rank}(\mathbf{C}(\mu)\mathbf{B}) = \text{rank}(\mathbf{B})$ (if such a selection exists).

Algorithm 1 Greedy Selection for Matching Condition

Notation:$\mu \cup \{i\}$ indicates setting the i^{th} element of μ to 1.

Input: Sensor matrix $\mathbf{C} \in \mathbb{R}^{p\times n}$, input matrix $\mathbf{B} \in \mathbb{R}^{n\times m}$, and a vector $\mathbf{b} \in \mathbb{R}^{p}_{\geq 0}$ indicating the cost of each row of $\mathbf{C}$.

Output: A sensor selection vector $\mu \in \{0,1\}^p$ that minimizes $\mathbf{b}'\mu$ while ensuring $\text{rank}(\mathbf{C}(\mu)\mathbf{B}) = \text{rank}(\mathbf{B})$.

```
Sort the rows of C to be in nondecreasing order by cost.
Initialize μ to the zero vector and i = 1
while rank(C(μ)B) < rank(B) do
    if rank(C(μ ∪ {i})B) > rank(C(μ)B) then
        μ = μ ∪ {i}
    end if
    i = i + 1
end while
return μ
```

Note that the sensor costs are allowed to be arbitrary nonnegative values in the above result. Thus, this captures (as a special case) the scenario where we already have a set of sensors installed on the system (e.g., to provide strong detectability), and we only need to select an additional set of sensors in order to satisfy the matching condition. Specifically, by setting the cost of all sensors that are already installed to be zero, the algorithm is guaranteed to select from the installed set of sensors first (as it checks the sensors in nondecreasing order of cost), and then to select the lowest cost subset of additional sensors to install. This is encapsulated in the following corollary.

Corollary 1 *Consider a linear system of the form* (2.1)*. Suppose we are given a set of p sensors $\mathcal{S}$, where each sensor in the set provides a scalar measurement of the form* (4.1)*. Let $\mathbf{C}$ be the matrix whose rows contain the measurement matrices of the sensors, and let $\mathbf{b} \in \mathbb{R}^p_{\geq 0}$ indicate the cost of each sensor. Suppose that some subset of the sensors in $\mathcal{S}$ is already installed on the system. Then, the lowest cost set of additional sensors to install so that the set of all installed sensors satisfies the matching condition can be found in polynomial time.*

5. THE STRONG DETECTABILITY SENSOR ATTACK PROBLEM

Having characterized the complexity of the sensor selection problem, we now turn our attention to the problem of *attacking* a set of deployed sensors in order to *remove* the property of strong detectability. We formulate this problem next, and then characterize its complexity.

5.1 Problem Formulation

Consider again system (2.1), and suppose that there are sensors deployed on the system (i.e., the output equation (2.2) is initially given) such that the system is strongly detectable. Instead of adding sensors to the system suppose one (i.e., an attacker) desires to remove sensors. The cost vector $\mathbf{v} \in \mathbb{R}^p_{\geq 0}$ assigns a nonnegative removal cost for each sensor. In other words, the i^{th} element of $\mathbf{v}$ denotes the cost of removing the i^{th} row from the matrix $\mathbf{C}$ for each $1 \leq i \leq p$.

As before, given an indicator vector $\mu \in \{0,1\}^p$, we denote $\mathbf{C}(\mu)$ to be the submatrix of $\mathbf{C}$ consisting of the rows corresponding to the sensors indicated by μ. Furthermore, the indicator vector $\mu^c \in \{0,1\}^p$ is the complement of μ (i.e. a '1' in μ is denoted as a '0' in μ^c and vice versa).

We consider the following problem.

Problem 3 (Strong Detectability Sensor Attack (SDSA)) *Suppose we are given the system matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the input matrix $\mathbf{B} \in \mathbb{R}^{n\times m}$, the output matrix $\mathbf{C} \in \mathbb{R}^{p\times n}$, and a cost vector $\mathbf{v} \in \mathbb{R}^p_{\geq 0}$. The Strong Detectability Sensor Attack Problem (SDSA) is to solve:*

$$\min_{\mu\in\{0,1\}^p} \mathbf{v}'\mu$$

$$s.t. \quad \exists\ |z_0| \geq 1 \qquad with\ \text{rank} \begin{bmatrix} \mathbf{A} - z_0\mathbf{I_n} & \mathbf{B} \\ \mathbf{C}(\mu^c) & \mathbf{0} \end{bmatrix} < n + m.$$

5.2 Complexity of SDSA

In this section, we will show that the SDSA problem is NP-hard. To do so, we will provide a reduction from the *MAX FLS*$^=$ problem, stated below.

Problem 4 (MAX FLS$^=$) *Consider a set of d homogeneous equations with f variables denoted by matrix* $\mathbf{T} \in \mathbb{R}^{d \times f}$*, and a nonnegative integer k.*
Question: *Is there a nonzero vector* $\mathbf{x} \in \mathbb{R}^f$ *such that at least k equalities in the equation* $\mathbf{Tx} = \mathbf{0}$ *are satisfied?*

The MAX FLS$^=$ problem is NP-hard [36]. We reduce MAX FLS$^=$ to SDSA, state a useful property of the created instance of SDSA, and finally prove that that the SDSA is NP-hard.

Polynomial-Time Reduction from MAX FLS$^=$ to SDSA

Given an instance of the MAX FLS$^=$ problem (with d equations and f variables denoted by matrix $\mathbf{T}$), we will create an instance of SDSA as follows. Define the matrices:

$$\mathbf{A} = \mathbf{0}_{f \times f}, \ \ \mathbf{B} = \mathbf{I}_f, \ \text{ and } \ \mathbf{C} = \mathbf{T}. \tag{5.1}$$

The cost vector $\mathbf{v}$ will consist of d elements, all equal to '1'.

Properties of the Created Instance

Lemma 4 *Given system* (2.1)-(2.2) *with* $\mathbf{A}$*,* $\mathbf{B}$ *and* $\mathbf{C}$ *defined in* (5.1)*, consider a sensor selection vector* $\nu \in \{0,1\}^d$*. The tuple* $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\nu))$ *has at least one unstable invariant zero if and only if* $\mathbf{C}(\nu)$ *is not full column rank.*

Proof Consider a sensor indicator vector $\nu \in \{0,1\}^d$ with associated matrix $\mathbf{C}(\nu)$. The tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\nu))$ will have an unstable invariant zero if and only if there is a complex number z_0 with $|z_0| \geq 1$, and a nonzero vector $\begin{bmatrix} \mathbf{X}_0' & \mathbf{U}_0' \end{bmatrix}'$ satisfying:

$$\begin{bmatrix} \mathbf{A} - z_0\mathbf{I}_f & \mathbf{B} \\ \mathbf{C}(\nu) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{X_0} \\ \mathbf{U_0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{5.2}$$

Substituting (5.1) into (5.2) we obtain:

$$z_0\mathbf{X_0} = \mathbf{U}_0 \tag{5.3}$$

$$\mathbf{C}(\nu)\mathbf{X_0} = \mathbf{0}. \tag{5.4}$$

If $\mathbf{C}(\nu)$ is not full column rank, there will exist a $\mathbf{X_0}$ vector in the null space of $\mathbf{C}(\nu)$, thereby satisfying (5.4). This $\mathbf{X_0}$ vector can be paired with any unstable z_0 value to form the vector $\mathbf{U}_0$ in (5.3).

On the other hand, if $\mathbf{C}(\nu)$ is full column rank, then the only solution to (5.4) is $\mathbf{X}_0 = \mathbf{0}$, and thus, from (5.3), $\mathbf{U}_0 = \mathbf{0}$. In this case, the matrix pencil in (5.2) has no nontrivial nullspace for any z_0, and thus has no invariant zeros.

Therefore, the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\nu))$ has an unstable invariant zero if and only if $\mathbf{C}(\nu)$ is not full column rank. ■

NP-hardness of SDSA

Using the reduction from MAX FLS$^=$ given by the system (5.1), and cost vector $\mathbf{v}$ consisting of all 1's, along with the property of such instances given above, we obtain the following result.

Theorem 5.2.1 *Given an instance of MAX FLS$^=$ (with a $d \times f$ matrix $\mathbf{T}$ and integer k) and the associated instance of SDSA (given by* (5.1)*), it is possible to remove $d-k$ or fewer sensors from $\mathbf{C}$ to make the resulting system no longer strongly detectable if and only if the answer to the instance of MAX FLS$^=$ is "yes". Thus, SDSA is NP-hard.*

Proof Suppose the answer to the instance of MAX FLS$^=$ is "yes" (i.e., there is a nonzero vector $\mathbf{x}$ satisfying at least k of the equalities in the equation $\mathbf{Tx} = \mathbf{0}$). Let μ^c be the indicator vector that selects the k satisfied equations from $\mathbf{Tx} = \mathbf{0}$. Consequently since $\mathbf{C} = \mathbf{T}$ in the created instance of SDSA (given by (5.1)), there must exist some nonzero vector $\mathbf{x}$ such that $\mathbf{C}(\mu^c)\mathbf{x} = \mathbf{0}$. Thus $\mathbf{x}$ is in the null space of $\mathbf{C}(\mu^c)$ and $\mathbf{C}(\mu^c)$ is not full rank. By Lemma 4, the tuple $(\mathbf{A}, \mathbf{B}, \mathbf{C}(\mu^c))$ is not strongly detectable if $\mathbf{C}(\mu^c)$ is not full rank. Thus, the conjugate μ of μ^c represents the sensors that once removed cause the system to lose strong detectability. Since at most $d - k$ sensors must be removed and each sensor has a cost of '1', there is a solution to the created SDSA instance that has cost at most $d - k$.

Now suppose that the answer to MAX FLS$^=$ is "no" (i.e., there are fewer than k equalities in $\mathbf{Tx} = \mathbf{0}$ that can be simultaneously satisfied). Therefore there must be more than $d - k$ sensors that must be removed for $\mathbf{C}$ to lose rank. By only removing $d-k$ sensors the system will remain strongly detectable. Thus, we see that given any instance of the MAX FLS$^=$ problem, we can create an instance of SDSA in polynomial-time, and solve the MAX FLS$^=$ instance by solving the sensor attack instance. Since MAX FLS$^=$ is NP-hard, SDSA is NP-hard. ■

Additionally this result indicates that it is NP-hard to minimally break the matching condition (2.9). This condition is satisfied if $\text{rank}(\mathbf{C}(\mu^c)\mathbf{B}) = \text{rank}(\mathbf{B})$. In the instance where $\mathbf{B}$ is full rank the only way for condition (2.9) to hold is for the $\mathbf{C}(\mu^c)$ matrix to be full column rank as well. Therefore, once again, the task is to remove the minimal amount of sensors from $\mathbf{C}$ such that it loses full column rank. Thus, as a positive result, it is NP-hard for an attacker to optimally select sensors to remove to break the matching condition (in contrast to the problem of selecting sensors to satisfy the matching condition, as shown in Corollary 1).

6. STRONG DETECTABILITY SENSOR SELECTION PROBLEM APPROXIMATION

When dealing with NP-hard problems, it is of interest to find polynomial-time *approximation algorithms* which provide solutions within a certain factor of the optimal. The following result provides a bound on the ability to approximate the minimum sensor cost in polynomial time.

Theorem 6.0.1 *For all $\epsilon > 0$, SDSS cannot be approximated within a factor of $(1-\epsilon)\log n$ where n is the number of states, unless $P = NP$.*

Proof By contradiction, suppose there is some $\epsilon > 0$ and an approximation algorithm for SDSS that always finds a set of sensors within a factor of $(1-\epsilon)\log n$ of the minimum cost. By running this algorithm on the constructed instance of SDSS given by (4.2), (4.3) and $\mathbf{b}$, we would obtain a set of sensors that provide strong detectability with a cost $\mathcal{B}$ that is within a factor $(1-\epsilon)\log n$ of the optimal cost. However the optimal cost is precisely equal to the smallest size of a set cover (by construction), and since the set of sensors yielded by the algorithm must be a set cover (by Lemma 2), we see that the algorithm would yield an approximation to Set Cover as well. Since Set Cover cannot be approximated within a factor of $(1-\epsilon)\log n$ of the optimal solution for any $\epsilon > 0$ (unless $P = NP$) [37], the result follows. ■

Consequently, the lower bound for the approximation ratio of SDSS is $(1-\epsilon)\log(n)$. We will next prove that $\log(n) + \log(m+1)$ is the upper bound of the SDSS problem (i.e., it can be achieved under certain conditions).

6.1 Constant Factor Approximation

Lemma 5 *A zero delay UIO (i.e.* $\text{rank}(\mathbf{CB}) = \text{rank}(\mathbf{B})$*) with* $\mathbf{D} = \mathbf{0}$ *is possible if and only if:*

$$\text{rank}([\mathbf{O}\ \mathbf{J}]_n) = m(n-1) + n. \tag{6.1}$$

Proof Suppose a zero delay UIO exisits. Since each $\mathbf{CB}$ block of the $\mathbf{J_n}$ matrix is full column rank and the last "m" columns are rank zero because $\mathbf{D} = \mathbf{0}$ the $\mathbf{J_n}$ matrix will have rank $m(n-1)$. The columns of the $\mathbf{O_n}$ matrix must be linearly independent from the columns of the $\mathbf{J_n}$ matrix. Thus, the $\mathbf{O_n}$ matrix will add rank n to the rank of the $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ matrix. The total rank of the $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ matrix is then $m(n-1)+n$.

Next, we will prove Lemma 5 in the other direction. Suppose the dimension of $[\mathbf{O}\ \mathbf{J}]_n$ is $m(n-1)+n$. Let $[\mathbf{O}\ \mathbf{J}]_v$ represent the matrix $[\mathbf{O}\ \mathbf{J}]_n$ without the last m columns. Since $\mathbf{D} = \mathbf{0}$ every column of $[\mathbf{O}\ \mathbf{J}]_v$ must be linearly independent for $[\mathbf{O}\ \mathbf{J}]_n$ to have dimension $m(n-1)+n$. The first $p(n-1)$ rows of the last m columns of $[\mathbf{O}\ \mathbf{J}]_v$ are empty. The last p rows and m columns of $[\mathbf{O}\ \mathbf{J}]_v$ form $\mathbf{CB}$. $\mathbf{CB}$ must be full rank for the last m columns of $[\mathbf{O}\ \mathbf{J}]_v$ to be linearly independent. ■

Lemma 6 *Consider a collection of row vector sets, where each row vector has length* q*. The minimum number of sets required to create a full column rank matrix* $\mathbf{V}$ *can be approximated within a factor of* $\log(q)$ *of the optimal number* k*.*

Proof The following is an adaption of the proof found in [38]. Suppose $\mathbf{V}$ must have rank q to be full column rank. Consider a greedy algorithm that iteratively selects the set that increases the rank of $\mathbf{V}$ the most. On the first iteration of the algorithm, there exists a set that will increase the rank of $\mathbf{V}$ by at least rank $\frac{q}{k}$. The dimension of the null space of $\mathbf{V}$ is now at most $q(1-\frac{1}{k})$. There is no guarantee that a set from the optimal collection of sets was chosen. On the second iteration of the algorithm there exists at sensor that will increase the rank at by least $\frac{q(1-\frac{1}{k})}{k}$ (i.e., the nullspace

over k). After each iteration j of the algorithm, $q(1-\frac{1}{k})^j$ dimensions will remain in the nullspace of $\mathbf{V}$. Thus, the only guarantee is that the algorithm will select on iteration $j+1$ a set that will increase the rank of $\mathbf{V}$ by at least $\frac{q(1-\frac{1}{k})^j}{k}$. After $k\log(q)$ iterations, the null space has dimension less than 1 since $q(1-\frac{1}{k})^{k\log q} < q(\frac{1}{e})^{\log(q)} = 1$. Once the null space of $\mathbf{V}$ is less than '1', the null space of $\mathbf{V}$ is empty and $\mathbf{V}$ is full rank. ■

Theorem 6.1.1 *The minimal number of sensors required for a zero delay UIO can be approximated within a factor of* $\log(n)+\log(m+1)$ *of the optimal.*

Proof From Lemma 5 we know that the $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ matrix must have rank $m(n-1)+n$. Since the last m columns of $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ remain zero they can be removed. There are exactly $m(n-1)+n$ columns of $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ remaining. Therefore, for a UIO to exist, $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ must be full column rank. Each sensor selected contributes a set of rows to $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$. As proven in Lemma 6 the sensors that will create a full rank $[\mathbf{O}\ \mathbf{J}]_\mathbf{n}$ matrix can be selected greedily with a guarantee of the answer being within $\log(m(n-1)+n)$ or on the order of $\log(n)+\log(m+1)$ of the optimal.

■

6.2 Synergies

Another greedy approach is to select the sensor that increases the difference between rank($[\mathbf{O}\ \mathbf{J}]$) and rank($\mathbf{J}$) the greatest. However, the output of this approach is complicated by the fact that an interesting synergistic effect between the sensors can occur. By synergy we imply the combined value of two or more sensors is greater than the sum of each sensor's value individually.

For instance, consider the following example:

$$\mathbf{A}=\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix},\mathbf{B}=\begin{bmatrix}1\\0\\0\end{bmatrix},\mathbf{C_1}=\begin{bmatrix}1&0&0\end{bmatrix},\mathbf{C_2}=\begin{bmatrix}1&1&0\end{bmatrix}.$$

In case 1 only sensor $\mathbf{C_1}$ is selected:

$$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}, \quad \mathbf{J} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \text{ and } \operatorname{rank}([\mathbf{O}\ \mathbf{J}]) - \operatorname{rank}(\mathbf{J}) = 3 - 2 = 1.$$

In case 2 only sensor $\mathbf{C_2}$ is selected:

$$\mathbf{O} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 2 \\ 1 & 3 & 5 \end{bmatrix}, \quad \mathbf{J} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \text{ and } \operatorname{rank}([\mathbf{O}\ \mathbf{J}]) - \operatorname{rank}(\mathbf{J}) = 3 - 2 = 1.$$

In case 3 both $\mathbf{C_1}$ and $\mathbf{C_2}$ are selected:

$$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 5 \end{bmatrix}, \quad \mathbf{J} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \text{ and } \operatorname{rank}([\mathbf{O}\ \mathbf{J}]) - \operatorname{rank}(\mathbf{J}) = 6 - 3 = 3.$$

Individually $\mathbf{C_1}$ and $\mathbf{C_2}$ increase the difference between rank($[\mathbf{O}\ \mathbf{J}]$) and rank($\mathbf{J}$) by only '1'. Yet, the combined effect of selecting $\mathbf{C_1}$ and $\mathbf{C_2}$ is that the difference between rank($[\mathbf{O}\ \mathbf{J}]$) and rank($\mathbf{J}$) is not '2' but '3'. Due to this phenomenon, the approximation bound for this greedy approach does not follow Theorem 6.1.1. In Lemma 6 the assumption is that the sensor that increases the rank the greatest is selected. Due to the synergistic effect in this algorithm the sensor that will increase the rank the greatest is not known. Depending on different combinations of sensors a sensor can cause a greater overall increase of rank. As of yet, no upper approximation bound has been found for this algorithm. However we see that this algorithm performs well in practice (in the next section).

7. SIMULATIONS

The NP-hardness reduction of the SDSS problem was based upon Set Cover. The paper [29] list the optimal solutions to a set of Set Cover instances. From these instances we created instances of SDSS based upon the SDSS NP-hardness reduction. The first 10 instances numbered $4.1 - 4.10$ contain 400 states, 200 inputs and 1200 possible sensors. The second 10 instances numbered $5.1 - 5.10$ contain 400 states 200 inputs and 2200 possible sensors. The last 5 instances numbered $6.1 - 6.5$ contain 400 states, 200 inputs and 1200 possible sensors. Instances in group 4 and 5 are built from instances of Set Cover with 2% density (i.e., each element is in 2% of the sets). Whereas instances in group 6 are built from instances of Set Cover with 5% density.

The code of each simulation was written in MATLAB and is displayed in Appendix A. We used the Purdue Halstead cluster to run the programs on one node and 20 cores running in parallel. The node had two Kaby Lake CPUs running at 2.60 GHz and 128 GB of memory.

Greedy 1 represents an algorithm that selects sensors to increase the rank of $[\mathbf{O}\ \mathbf{J}]$ to $m(n-1)+n$. Greedy 2 represents an algorithm that selects sensors to increase the difference of $\text{rank}([\mathbf{O}\ \mathbf{J}]) - \text{rank}(\mathbf{J})$ to equal 'n'. In both Greedy 1 and Greedy 2 the most cost effective (i.e., $\frac{\text{rank increase}}{\text{cost}}$) sensor is selected. Each instance of SDSS is formed by reduction in Section 4. The reduction guarantees that $\text{rank}(\mathbf{CB}) = \text{rank}(\mathbf{B})$ for all instances. Consequently the results of Greedy 1 and Greedy 2 both form zero-delay observers.

In the following tables, the ratio between the computed value by each algorithm and the optimal value of each instance is recorded. In every problem set, Greedy 2 outperforms Greedy 1. Although we have a approximation guarantee for Greedy 2 of $\log(n) + \log(m+1)$, Greedy 1 is outperformed by Greedy 2. In theory these two algorithms could be run side by side and the algorithm that provides the lowest valued

Table 7.1.
SDSS Approximation Over Problem Set 4: n = 400, m = 200, p = 1200, and 2% density

Problem		Greedy 1		Greedy 2	
number	optimal value	value	ratio	value	ratio
4.1	429	1848	4.31	463	1.08
4.2	512	2110	4.12	582	1.14
4.3	516	2097	4.06	598	1.16
4.4	494	1950	3.95	548	1.11
4.5	512	2006	3.92	577	1.13
4.6	560	2194	3.92	615	1.10
4.7	430	1859	4.32	476	1.11
4.8	492	2281	4.64	533	1.08
4.9	641	2316	3.61	747	1.17
4.10	514	1982	3.86	556	1.08

solution can be selected. By doing so, the user is guaranteed a $\log(n) + \log(m+1)$ approximation bound when Greedy 2 may not approximate within $\log(n)+\log(m+1)$ of the optimal solution. The time required to solve each problem was roughly 1000 - 2000 seconds on the Purdue Halstead clusters.

Table 7.2.
SDSS Approximation Over Problem Set 5: n = 400, m = 200, p = 2200, and 2% density

Problem		Greedy 1		Greedy 2	
number	optimal value	value	ratio	value	ratio
5.1	253	1096	4.33	289	1.14
5.2	302	1177	3.90	348	1.15
5.3	226	991	4.39	246	1.09
5.4	242	956	3.95	265	1.10
5.5	211	967	4.58	236	1.12
5.6	213	990	4.65	251	1.18
5.7	293	1211	4.13	326	1.11
5.8	288	1052	3.65	323	1.12
5.9	279	1106	3.96	312	1.12
5.10	265	1125	4.25	289	1.09

Table 7.3.
SDSS Approximation Over Problem Set 6: n = 400, m = 200, p = 1200, and 5% density

Problem		Greedy 1		Greedy 2	
number	optimal value	value	ratio	value	ratio
6.1	138	1811	13.12	159	1.15
6.2	146	2046	14.01	170	1.16
6.3	145	2047	14.12	161	1.11
6.4	131	1892	14.44	149	1.14
6.5	161	1973	12.25	196	1.22

8. SUMMARY AND FUTURE WORK

In this thesis, we showed that it is NP-hard to select a set of sensors of minimum cost in order to make a system strongly detectable. Our proof shows that this result holds even for stable systems, and thus the computational complexity arises from the effects of the unknown inputs in the system, as opposed to the need to ensure system detectability. We also showed that it is not possible to approximate the minimum cost within a factor that is logarithmic in the size of the problem. However, we showed that if a set of sensors has already been chosen to make a system strongly detectable, finding an additional set of sensors of minimum cost in order to obtain zero-delay estimation can be done in polynomial time. Next, we considered the problem of attacking a given strongly detectable system by removing a set of sensors to remove the strong detectability property. We showed that this problem is also NP-hard. Lastly, we provided simulation results indicating that Greedy 2 outperformed Greedy 1 although Greedy 2 does not have an upper bound for guaranteed performance. There are a variety of avenues for future research, including determining instances of the sensor selection and attack problems where optimal (or near-optimal) solutions can be found in polynomial time.

REFERENCES

REFERENCES

[1] F. Menolascina, V. Siciliano, and D. Di Bernardo, "Engineering and control of biological systems: a new way to tackle complex diseases," *FEBS letters*, vol. 586, no. 15, pp. 2122–2128, 2012.

[2] L. Figueiredo, I. Jesus, J. T. Machado, J. R. Ferreira, and J. M. De Carvalho, "Towards the development of intelligent transportation systems," pp. 1206–1211, 2001.

[3] A. Chakrabortty and M. D. Ilić, "Control and optimization methods for electric smart grids," vol. 3, 2011.

[4] G. Bumiller, L. Lampe, and H. Hrasnica, "Power line communication networks for large-scale control and automation systems," *IEEE Communications Magazine*, vol. 48, no. 4, 2010.

[5] T. H. Summers, F. L. Cortesi, and J. Lygeros, "On submodularity and controllability in complex dynamical networks," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 1, pp. 91–101, 2016.

[6] V. Gupta, T. H. Chung, B. Hassibi, and R. M. Murray, "On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage," *Automatica*, vol. 42, no. 2, pp. 251–260, 2006.

[7] S. T. Jawaid and S. L. Smith, "Submodularity and greedy algorithms in sensor scheduling for linear dynamical systems," *Automatica*, vol. 61, pp. 282–288, 2015.

[8] M. P. Vitus, W. Zhang, A. Abate, J. Hu, and C. J. Tomlin, "On efficient sensor scheduling for linear dynamical systems," *Automatica*, vol. 48, no. 10, pp. 2482–2493, 2012.

[9] H. Rowaihy, S. Eswaran, M. Johnson, D. Verma, A. Bar-Noy, T. Brown, and T. La Porta, "A survey of sensor selection schemes in wireless sensor networks," vol. 6562, p. 65621A, 2007.

[10] A. Olshevsky, "Minimal controllability problems," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 3, pp. 249–258, 2014.

[11] M. Van De Wal and B. De Jager, "A review of methods for input/output selection," *Automatica*, vol. 37, no. 4, pp. 487–510, 2001.

[12] X. Liu, B. M. Chen, and Z. Lin, "On the problem of general structural assignments of linear systems through sensor/actuator selection," *Automatica*, vol. 39, no. 2, pp. 233–241, 2003.

[13] H. Zhang, R. Ayoub, and S. Sundaram, "Sensor selection for Kalman filtering of linear dynamical systems: Complexity, limitations and greedy algorithms," *Automatica*, vol. 78, pp. 202–210, 2017.

[14] L. Ye, S. Roy, and S. Sundaram, "On the complexity and approximability of optimal sensor selection for Kalman filtering," pp. 5049–5054, 2018.

[15] V. Tzoumas, A. Jadbabaie, and G. J. Pappas, "Sensor placement for optimal Kalman filtering: Fundamental limits, submodularity, and algorithms," pp. 191–196, 2016.

[16] S. Pequito, G. Ramos, S. Kar, A. P. Aguiar, and J. Ramos, "The robust minimal controllability problem," *Automatica*, vol. 82, pp. 261–268, 2017.

[17] M.-A. Belabbas, "Geometric methods for optimal sensor design," vol. 472, no. 2185, p. 20150312, 2016.

[18] R. J. Patton and J. Chen, "Robust model-based fault diagnosis for dynamic systems," 1999.

[19] S. Sundaram and C. N. Hadjicostis, "Distributed function calculation via linear iterative strategies in the presence of malicious agents," *IEEE Trans on Automatic Control*, vol. 56, no. 7, pp. 1495–1508, 2011.

[20] F. Pasqualetti, F. Dörfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2715–2729, 2013.

[21] A. Teixeira, K. C. Sou, H. Sandberg, and K. H. Johansson, "Secure control systems: A quantitative risk management approach," *IEEE Control Systems*, vol. 35, no. 1, pp. 24–45, 2015.

[22] S. Roy and R. Dhal, "Situational awareness for dynamical network processes using incidental measurements," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 2, pp. 304–316, 2015.

[23] S. Sharifi, D. Krishnaswamy, and T. Š. Rosing, "Prometheus: A proactive method for thermal management of heterogeneous MPSoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 7, pp. 1110–1123, 2013.

[24] Y. Guan and M. Saif, "A novel approach to the design of unknown input observers," *IEEE Transactions on Automatic Control*, vol. 36, no. 5, pp. 632–635, 1991.

[25] J. Chen, R. J. Patton, and H.-Y. Zhang, "Design of unknown input observers and robust fault detection filters," *International Journal of control*, vol. 63, no. 1, pp. 85–105, 1996.

[26] M. Saif and Y. Guan, "A new approach to robust fault detection and identification," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 3, pp. 685–695, 1993.

[27] M. L. Hautus, "Strong detectability and observers," *Linear Algebra and its applications*, vol. 50, pp. 353–368, 1983.

[28] S. Sundaram and C. N. Hadjicostis, "Delayed observers for linear systems with unknown inputs," *IEEE Transactions on Automatic Control*, vol. 52, no. 2, pp. 334–339, 2007.

[29] J. E. Beasley, "A lagrangian heuristic for set-covering problem," *Naval Research Logistics (NRL)*, vol. 37, no. 1, pp. 151 – 164, 1990.

[30] M. Rangarajan, T. M. Lambert, and A. C. Wynn, "System and method of managing heat in multiple central processing units," Patent, Jan. 26, 2010.

[31] L. Garrett, N. Jarboe, D. J. Patton, and L. L. Mullins, "The effects of encapsulation on dismounted warrior performance," Army Research Lab Aberdeen Proving Ground MD Human Research And Engineering Directorate, Tech. Rep., 2006.

[32] S. Shreyas, "Fault-tolerant and secure control systems," June 2017.

[33] W. Kratz, "Characterization of strong observability and construction of an observer," *Linear algebra and its applications*, vol. 221, pp. 31–40, 1995.

[34] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[35] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[36] E. Amaldi and V. Kann, "The complexity and approximability of finding maximum feasible subsystems of linear relations," *Theoretical computer science*, vol. 147, no. 1-2, pp. 181–210, 1995.

[37] I. Dinur and D. Steurer, "Analytical approach to parallel repetition," pp. 624–633, 2014.

[38] S. Dasgupta, C. Papadimitriou, and U. Vazirani, "Algorithms," pp. 152–154, 2008.

APPENDIX

A. CODE

The following is the MATLAB code used for parsing the data files from [29].

```
function [A,B,C,cost] = Parsing(filename)
    fid = fopen(filename);

    line_1 = double(str2num(fgetl(fid)));
    p = line_1(1);
    n = line_1(2);
    S = zeros(p,n);

    cost = zeros(1,n+p);
    size(cost)
    indx = p+1;

    for i = 1:ceil(n/15)-1
        cost(indx:indx+14) = double(str2num(fgetl(fid)));
        indx = indx+15;
    end

    cost(indx:end) = double(str2num(fgetl(fid)));

    for j = 1:p
        tot = double(str2num(fgetl(fid)));
        for i =1:ceil(tot/15)
            line = double(str2num(fgetl(fid)));
            for i = 1:size(line,2)
                S(j,line(i)) = 1;
            end
        end
```

```
28      end
29
30      S = S';
31
32      n_new = n;
33      p_new = p;
34
35      n = p_new*2;
36      p = n_new*2;
37
38      A1 = zeros(n/2,n/2);
39      A2 = diag(linspace(1,n/2,n/2));
40      A3 = zeros(n/2,n/2);
41      A4 = zeros(n/2,n/2);
42
43      A = [A1,A2;A3,A4];
44
45      B1 = eye(n/2);
46      B2 = -eye(n/2);
47
48      B = [B1;B2];
49
50      C1 = eye(n/2,n/2);
51
52      C2 = zeros(n/2,n/2);
53
54      C3 = zeros(p/2,n/2);
55
56      C4 = S;
57
58
59      C = [C1,C2;C3,C4];
```

The following code was used to compute Greedy 1.

```
parpool(20);

number = [4.1,4.2,4.3,4.4,4.5];

for name = number
    filename = strcat('SCP',name,'.txt');
    [A,B,C,cost] = Parsing(filename);
    tic
    A = sparse(A);
    B = sparse(B);
    C = sparse(C);

    n = size(A,1);
    p = size(C,1);
    m = size(B,2);

    num = zeros(1,n-1);
    num_tot = zeros(1,n-1);

    parfor x = 1:n-1
        mat = C*A^(n-x-1)*B;
        mat2 = C*A^(n-x);
        [i,~,~] = find(mat);
        [i2,~,~] = find(mat2);
        num(x) = size(i,1);
        num_tot(x) = size(i,1)*(x);
        num2(x) = size(i2,1);
    end

    x = n;
    mat2 = C*A^(n-x);
    [i2,~,~] = find(mat2);
    num2(x) = size(i2,1);

```

```
35      tot = sum(num_tot);
36      tot2 = sum(num2);
37
38
39      I = zeros(1,tot);
40      J = zeros(1,tot);
41      V = zeros(1,tot);
42
43      I2 = zeros(1,tot2);
44      J2 = zeros(1,tot2);
45      V2 = zeros(1,tot2);
46
47      indx = 1;
48      indx2 = 1;
49
50      for x = 1:n-1
51          mat = C*A^(n-x-1)*B;
52          mat2 = C*A^(n-x);
53
54          [i,j,v] = find(mat);
55          [i2,j2,v2] = find(mat2);
56
57          i = i+(n-x)*p;
58          i2 = i2+(n-x)*p;
59
60          I2(indx2:indx2+num2(x)-1) = i2;
61          J2(indx2:indx2+num2(x)-1) = j2;
62          V2(indx2:indx2+num2(x)-1) = v2;
63          indx2 = indx2+num2(x);
64
65          for y = 1:x
66              I(indx:indx+num(x)-1) = i+p*(y-1);
67              J(indx:indx+num(x)-1) = j+m*(y-1);
68              V(indx:indx+num(x)-1) = v;
69              indx = indx+num(x);
```

```
70          end
71      end
72
73      x = n;
74      mat2 = C*A^(n-x);
75      [i2,j2,v2] = find(mat2);
76      i2 = i2+(n-x)*p;
77      I2(indx2:indx2+num2(x)-1) = i2;
78      J2(indx2:indx2+num2(x)-1) = j2;
79      V2(indx2:indx2+num2(x)-1) = v2;
80      indx2 = indx2+num2(x);
81
82      O = sparse(I2,J2,V2,p*n,n);
83      J = sparse(I,J,V,p*n,m*n);
84      OJ = [O,J];
85      r = 0;
86      k = 0;
87      C_pick = zeros(1,n/2);
88
89      indx = 1;
90      for i = 1:n/2
91          C_pick(indx:indx+n/2-1) =
92          linspace(1,n/2,n/2)+ones(1,n/2)*p*(i-1);
93          indx = indx+n/2;
94      end
95      pick = linspace(1,n/2,n/2);
96      T = 0;
97
98      rold = sprank(OJ(C_pick,:));
99      rinc = zeros(1,p);
100
101     while r < n
102         parfor z = n/2:p
103             C_test = [C_pick, linspace(z,p*n-p+z,n)];
104             H1 = OJ(C_test,:);
```

```
            H2 = J(C_test,:);
            rnew(z) = sprank(H1) - sprank(H2);
            rinc(z) = (rnew(z)-rold)/cost(z);
        end
        rinc(pick) = 0;
        [~,argmax] = max(rinc);
        r = rnew(argmax);
        rold = rnew(argmax);
        pick = [pick, argmax];
        C_pick = [C_pick,linspace(argmax,p*n-p+argmax,n)];
        rinc = zeros(1,p);
    end

    goal = sum(cost(pick));
    fprintf('For run %s: %d\n',filename,goal)
    toc
end
```

The following code was used to compute Greedy 2.

```
parpool(20);

number = [4.1,4.2,4.3,4.4,4.5];

for name = number
    filename = strcat('SCP',name,'.txt');
    [A,B,C,cost] = Parsing(filename);
    tic
    A = sparse(A);
    B = sparse(B);
    C = sparse(C);

    n = size(A,1);
    p = size(C,1);
```

```
15      m = size(B,2);
16
17      num = zeros(1,n-1);
18      num_tot = zeros(1,n-1);
19
20      parfor x = 1:n-1
21          mat = C*A^(n-x-1)*B;
22          mat2 = C*A^(n-x);
23          [i,~,~] = find(mat);
24          [i2,~,~] = find(mat2);
25          num(x) = size(i,1);
26          num_tot(x) = size(i,1)*(x);
27          num2(x) = size(i2,1);
28      end
29
30      x = n;
31      mat2 = C*A^(n-x);
32      [i2,~,~] = find(mat2);
33      num2(x) = size(i2,1);
34
35      tot = sum(num_tot);
36      tot2 = sum(num2);
37
38
39      I = zeros(1,tot);
40      J = zeros(1,tot);
41      V = zeros(1,tot);
42
43      I2 = zeros(1,tot2);
44      J2 = zeros(1,tot2);
45      V2 = zeros(1,tot2);
46
47      indx = 1;
48      indx2 = 1;
49
```

```
50      for x = 1:n-1
51          mat = C*A^(n-x-1)*B;
52          mat2 = C*A^(n-x);
53
54          [i,j,v] = find(mat);
55          [i2,j2,v2] = find(mat2);
56
57          i = i+(n-x)*p;
58          i2 = i2+(n-x)*p;
59
60          I2(indx2:indx2+num2(x)-1) = i2;
61          J2(indx2:indx2+num2(x)-1) = j2;
62          V2(indx2:indx2+num2(x)-1) = v2;
63          indx2 = indx2+num2(x);
64
65          for y = 1:x
66              I(indx:indx+num(x)-1) = i+p*(y-1);
67              J(indx:indx+num(x)-1) = j+m*(y-1);
68              V(indx:indx+num(x)-1) = v;
69              indx = indx+num(x);
70          end
71      end
72
73      x = n;
74      mat2 = C*A^(n-x);
75      [i2,j2,v2] = find(mat2);
76      i2 = i2+(n-x)*p;
77      I2(indx2:indx2+num2(x)-1) = i2;
78      J2(indx2:indx2+num2(x)-1) = j2;
79      V2(indx2:indx2+num2(x)-1) = v2;
80      indx2 = indx2+num2(x);
81
82      O = sparse(I2,J2,V2,p*n,n);
83      J = sparse(I,J,V,p*n,m*n);
84      OJ = [O,J];
```

```
85      r = 0;
86      k = 0;
87      C_pick = zeros(1,n/2);
88
89      indx = 1;
90      for i = 1:n/2
91          C_pick(indx:indx+n/2-1) =
92          linspace(1,n/2,n/2)+ones(1,n/2)*p*(i-1);
93          indx = indx+n/2;
94      end
95      pick = linspace(1,n/2,n/2);
96      T = 0;
97
98      rold = sprank(OJ(C_pick,:));
99      rinc = zeros(1,p);
100
101     while r < n*(m-1)+n
102         parfor z = n/2:p
103             C_test = [C_pick, linspace(z,p*n-p+z,n)];
104             H1 = OJ(C_test,:);
105             rnew(z) = sprank(H1);
106             rinc(z) = (rnew(z)-rold)/cost(z);
107         end
108         rinc(pick) = 0;
109         [~,argmax] = max(rinc);
110         r = rnew(argmax);
111         rold = rnew(argmax);
112         pick = [pick, argmax];
113         C_pick = [C_pick,linspace(argmax,p*n-p+argmax,n)];
114         rinc = zeros(1,p);
115     end
116
117     goal = sum(cost(pick));
118     fprintf('For run %s: %d\n',filename,goal)
119     toc
```

```
120 end
```

VITA

VITA

Nathaniel Woodford received the Bachelors of Science Degree in Astronautical Engineering from the United States Air Force Academy in 2017. He is currently a masters graduate student in the School of Electrical and Computer Engineering at Purdue University. His interests include fault-tolerant control, combinatorial optimization, and machine learning.