2017

# On minimal support solutions of underdetermined systems of linear equations

Darrin Thomas Rasberry
*Iowa State University*

# On minimal support solutions of underdetermined systems of linear equations

by

**Darrin Thomas Rasberry**

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Mathematics

Program of Study Committee:
Irvin Hentzel, Co-major Professor
Sung Song, Co-major Professor
Jonathan Smith
Arka Ghosh
Elgin Johnston

Iowa State University

Ames, Iowa

2017

# DEDICATION

*Σοί, Κύριε.*

# TABLE OF CONTENTS

# LIST OF FIGURES

# ACKNOWLEDGEMENTS

I would like to thank Dr. Irvin Hentzel and all of my graduate committee for their guidance, patience and support throughout my graduate career. I am glad to have this chance to renew and complete my path toward this dissertation and toward my degree.

I would also like to thank Ellsworth Community College, Des Moines Area Community College, and Mercy College of Health Sciences - my employers during my return to graduate school and during the writing of this thesis - for working with my schedule, deadlines, and research needs for the past three and a half years. I also thank all my students throughout this time for their encouragement and inspiration.

Finally, I would like to thank my loving wife, Mary; my brother Garland and my sister Ashley; my nieces Reese and Nylah; my aunts, uncles, and cousins; my step-families and all family by marriage; my father in faith Seraphim and my father in spirit Basil of St. George Greek Orthodox Church in Des Moines; all my friends who stuck by me through these difficult years; many loved ones still here in spirit and in heart, including my mother Brenda, my father Charlie, all my supportive grandparents, and my sister-in-law Michelle; and, most importantly, the only wise God, by Whom this work (and all things) were made.

# ABSTRACT

This paper explores the nature and application of minimal-support solutions of underdetermined systems of linear equations. First, methods for directly solving the problem are evaluated for effectiveness, and cases are shown to demonstrate that these direct methods are unreliable for finding minimal support solutions. The NP-Hardness of minimal-support solution recovery is then demonstrated over any field for the first time in the literature, and further NP-Hardness results are explored after this presentation. Following these expositions, a summary of current techniques in the practice of Compressive Sensing is given, and a novel method for comprehensively solving minimal-support solutions of underdetermined systems over any field is stated, discussed and proven. A summary of findings and avenues for future opportunities concludes the dissertation.

# CHAPTER 1. INTRODUCTION

## 1.1 Statement of the Problem and Clarification of Notation

The purpose of this brief introductory section is to state the dissertation problem clearly, to discuss assumptions used throughout the dissertation and to justify these assumptions, to clarify the meaning and application of conventional terminology, and to provide a layout for the remainder of the paper.

### 1.1.1 Problem Statement

This paper explores minimal support solutions of underdetermined systems of linear equations over any given field. Given a matrix $A \in \mathbb{F}^{m \times n}$ and a vector $b \in \mathbb{F}^m$, we seek solution vectors $x \in \mathbb{F}^n$ such that $Ax = b$ and such that the number of nonzeros of $x$ is minimal over all possible solutions to the system. A formal presentation of the meaning of underdetermined systems of linear equations, and a formal statement of goals generally sought in this work, are stated following the discussion below.

### 1.1.2 Universal Definitions and Assumptions

$\mathbb{F}$ is assumed to be any field, finite or infinite, unless specified. Vector spaces over $\mathbb{F}$ are finite-dimensional.

For ease of notation, we refer to the *weight*, or number of nonzeros in the vector $x$, as the $\ell_0$ *norm* of $x$; the evaluation of the $\ell_0$ norm applied to a particular vector $x$ is noted as $\|x\|_0$. It is necessary to clarify that $\|x\|_0$ is *not* a formal norm except in the case $\mathbb{F} = \mathbb{F}_2$, as homogeneity is broken when $x \neq 0$ and $a \neq 0, -1$, or 1. Nonetheless, we follow the literature in referring to this function as a norm.

The weight of the vector $(x + y)$ is no more than the weight of $x$ plus the weight of $y$. Since $x_k + y_k$ is nonzero if and only if at least one of $x_k$ and $y_k$ is not zero, this means $\|x+y\|_0 \leq \|x\|_0 + \|y\|_0$, meaning that the $\ell_0$ "norm" obeys the triangle inequality despite not being a formal norm.

The *support* of $x$ is the set $S$ of indices of $x$ such that $x_i \neq 0$ if and only if $i \in S$. The term *support* is sometimes used to denote the weight $\|x\|_0$ of the vector $x$ as above, which is the cardinality of the support set; the context should make the particular usage clear. If a vector $x$ is such that $\|x\|_0 \leq s$, the vector is known as *$s$-sparse.*

A system of linear equations $Ax = b$ is often denoted by the augmented matrix $[A|b]$. The augmented matrix representing the row-reduced echelon form (RREF) of $[A|b]$ is represented $RREF([A|b])$.

If dependent rows in $A$ are present, $RREF([A|b])$ contains zero rows, which are moved to the bottom of the tableau. Since zero rows provide no new information, we therefore often take the convention that $A$ has $m$ independent rows, though we attempt to address row-dependent matrices in all pertinent contexts in the implementation given in Chapter 4. The main algorithm of this dissertation, Algorithm 4.2.2, operates with systems already prepared in the format of having dependent rows identified and removed.

### 1.1.3 Formal Definitions and Formal Problem Statement

We formally define an underdetermined system of linear equations, and this definition is used throughout the paper except when context demands consideration of a more general form.

**Definition 1.1.** A consistent *underdetermined system of linear equations* $Ax = b$ over a field $\mathbb{F}$ is a constant matrix $A \in \mathbb{F}^{m \times n}$ of full rank, $m, n \in \mathbb{N}$, $m \leq n$, together with a constant nonzero vector $b \in \mathbb{F}^m$ and vector $x \in \mathbb{F}^n$.

The inequality in Definition 1 is usually taken to be strict throughout this work, as cases of equality indicate that $A$ is invertible, since $A$ is assumed to be of full rank. Additionally, $b$ is typically, but not always, assumed to be nonzero, since cases where $b = \mathbf{0}$ are also trivially answered with the unique solution of minimal support, $x = \mathbf{0}$.

A system of linear equations that is not consistent is *inconsistent.* A system of linear equations for which $n < m$ but that fits Definition 1.1 otherwise is called an *overdetermined* system; such systems, though interesting in themselves, are not the focus of this dissertation.

We now formally define the central problem of the dissertation.

**Problem 1.2.** Given an underdetermined system of linear equations $Ax = b$ in accordance with Definition 1.1, find the set of all solutions of minimal support.

The importance of Problem 1.2 is that, under suitable conditions, $m < n$ measurements can be taken on an $s$-sparse vector $x$, which can then be recovered from the smaller measurement vector $b \in \mathbb{F}^m$. This practice is known as *Compressive Sensing* (12). MRI scanning, radar, image compression, error correction, machine learning, automated proof solving, and rank minimization are some practical applications that can benefit from minimal-support solution recovery and methods that can efficiently approximate solutions to this problem.

Problem 1.2 may be classified in terms of the function *arg min,* which reports the actual objects that lead to the minimization of the measured quantity. In these terms, Problem 1.2 appears as

$$\arg \min_{x : Ax = b} \left( \|x\|_0 \right).$$

As an *optimization problem*, Problem 1.2 is restated as:

$$\text{minimize } \|x\|_0 \text{ subject to the constraint that } Ax = b.$$

Unfortunately, the optimization form of Problem 1.2 is not a *convex* optimization problem over any field. Consequences of this fact are given in the next chapter, and the proof of this fact rests on the proof that Problem 1.2 is in a class of problems known as *NP-Hard,* informally meaning that any task designed to recover a minimal support solution to a given system in accordance with Definition 1.1 must involve a number of steps whose size is not bounded by any function of the size of the input system.

For the system $Ax = b$, $A$ is sometimes referred as the *measurement matrix*, with the $m$ elements of $b$, the *measurement vector,* referred to as *measurements.* This terminology is used within the related discipline of Compressive Sensing, which is discussed at length in Chapter 4, and, as the problem addressed by this dissertation is intimately related to this discipline, these terms are used liberally throughout this paper.

## 1.2 Organization of the Dissertation

Basic terms and ideas from abstract and linear algebra used in this dissertation are defined in (1), (2), and (3). Analytic terminology can be referenced in (7) or (8). Combinatorial optimization and complexity presented in a precise mathematical format can be found in (4), which is used as a primary source in the development of Chapter 3. Background information and terminology about Compressive Sensing may be found in (12). Further terms and theory from other texts and research papers are cited within their relevant context.

Chapter 2 explores the failure of common direct approaches to solve the dissertation problem over general fields. First, an example is provided to show that solving the row-reduced echelon form representing the system $Ax = b$ for the dependent variables in a direct way fails to always attain a solution of minimal support. This is followed by the failure of approaching the problem by attempting to find a solution to $Ax = b$ that minimizes the metric $\|x\|_p$, $p \geq 1$, though the special case for $p = 1$ is mentioned as a candidate for succeeding in approximating Problem 1.2; to that end, a linear programming example is shown to generate minimal-support solutions. The chapter concludes with a discussion on attempting to approximate the solution through the metrics $\|x\|_q^q$, $0 < q < 1$, a discussion that concludes in Chapter 3.

Chapter 3 formalizes the NP-Hardness of Problem 1.2 over any field $\mathbb{F}$ for the first time in the literature through techniques in analysis of algorithms and *combinatorial optimization.* Following the introduction and definitions, each section tackles the combinatorial difficulty of Problem 1.2 from a different direction; the first section establishes the promised comprehensiveness of difficulty over any field, the second connects the problem with the related $\ell_q$, $0 < q < 1$-minimization technique discussed in Chapter 2. The third section proves the NP-Hardness of Problem 1.2 by reducing the Bounded Predecessor Existence Problem (BPEP) in automata theory to instances of the problem over the binary field.

Chapter 4 opens with a brief overview and discussion of basic theory in the recently developed discipline of *Compressive Sensing.* Several approximation techniques, such as Basis Pursuit and Orthogonal Matching Pursuit, are briefly given exposition, and a discussion of related bounds and conditions under which these approximations operate is given. The overview concludes with a program, $F^2OMP$, which shares some critical components of operation with the original algorithm presented in the second part of the chapter. Algorithm 4.2.2 is then presented in pseudocode (along with adjunct algorithms needed to prepare MinSup and its operation) as the formal, novel solution to Problem 1.2; a discussion outlining major aspects of the algorithm follows, and the dissertation concludes with an inductive proof that the algorithm both halts and properly performs its assigned task.

Chapter 5 highlights analytical needs and possible techniques that could improve Algorithm 4.2.2, first addressing some combinatorial aspects of the problem and then presenting opportunities for parallelization and approximation.

# CHAPTER 2. EVALUATION OF DIRECT APPROACHES

The purpose of this chapter is to present examples and observations clarifying the nature and difficulty of solving underdetermined systems of linear equations. We first show that direct methods do not necessarily yield minimal-support solutions. We then turn to evaluating the subtle approach utilizing the decomposition $\begin{bmatrix} A & -A \end{bmatrix} \begin{bmatrix} x^+ & x^- \end{bmatrix}$ for $x^+, x^- \geq 0$ via linear programming, with minimizing $\|x\|_1$ as the objective function.

Before concluding the chapter, we introduce another method which succeeds in approximating minimal support solutions. This proposed approximation method is dismissed as computationally ineffective in Chapter 3, following the introduction of terms and methods needed to address the capability of this approach.

In some cases that follow, we slightly abuse notation by refraining from noting the transpose of row vectors that should appear as column vectors for proper matrix multiplication. The context should make the usage of these vectors as column vectors obvious.

## 2.1 Failure of Direct Methods

### 2.1.1 Setting Independent Variables to Zero

In this section, a counterexample to row-reducing an augmented system representing an underdetermined system of equations is shown. The section leads with a simple, but important, observation on the bound for the number of nonzeros a minimal support solution is allowed to have. In the Observation below, and in the rest of the paper, let $e_i$ stand for the unit column vector with 1 in row $i$ and 0 in all other rows; in other words,

$e_1, \ldots, e_r$ are the first $r$ columns (in proper order) of the identity matrix. Context should dictate the dimension of these vectors whenever this notation is used.

**Observation 2.1.** $\min\limits_{x:Ax=b}(\|x\|_0) \leq m$.

*Proof.* Let $Ax = b$ be consistent, underdetermined, and of full rank.

Without loss of generality, let $RREF([A|b]) = [I_{m\times m} : A'|b']$; if $RREF[(A|b)]$ is not of this form, actions from the permutation group $S_n$ attain this form, so long as rows of $b'$ are permuted accordingly.

An immediate solution to the original system is, therefore, $x = \sum\limits_{j=1}^{m} b'_j e_j$, which is of weight bounded by $m$. ////

We informally refer to the solution indicated in Observation 2.1 as the *readoff solution* for a matrix in row-reduced echelon form. The next example demonstrates its failure.

**Example 2.2.** Let $F = F_3$, and define the augmented

$$[A|b] = \begin{bmatrix} 1 & 0 & 0 & 2 & 2 & 2 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

The solution proposed by simply setting the pivot (dependent) variables to the entries of $b$ and ignoring the free variable columns is $x_1 = 1, x_2 = 2, x_3 = 1$, yielding the solution vector $x = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 \end{bmatrix}$. We see immediately that this vector solves the linear system, and that $\|x\|_0 = 3$, but the vector $\begin{bmatrix} 0 & 0 & 1 & 2 & 0 & 0 \end{bmatrix}$ is also a solution, as

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + 2\begin{bmatrix} 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 4 \text{ (mod 3)} & 2 \text{ (mod 3)} & 1 \text{ (mod 3)} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}.$$

Hence, we conclude $\min_{x:Ax=b}(\|x\|_0) \leq 2 < 3 = m$ for this matrix, and the proposed method of setting independent variables to zero fails to retrieve a minimal support solution.

### 2.1.2 Shortest Distance Approximation

In this section, we analyze whether the shortest $\ell_2$ distance

$$\min_{x:Ax=b}(\|x\|_2)$$

over the solution space for $Ax = b$ leads to the minimal support solution. We begin with an observation that is direct from the construction in Definition 1.1.

**Observation 2.3.** The solution set for an underdetermined system of linear equations given in terms of Definition 1.1 is an Affine space of the form $x = x' + N$, where $x' \in F^n$ is a nonzero vector such that $Ax' = b$, and where $N$ is the null space of $A$.

*Proof.* Let $s \in N$ the null space of $A$. Then $A(x' + s) = A(x') + A(s) = A(x') + 0 = b$. Furthermore, let $Ax = b$. Then $x = x' + x - x'$. We have $Ax = A(x' + x - x') = Ax' + A(x - x') = b$, and since $A(x') = b$, $A(x - x') = 0$, implying $x - x' \in N$. ////

The next example dismisses a second obvious approach to finding minimal support solutions: minimizing the $\ell_2$ distance between the origin and the Affine space defining the solution set to a linear system $Ax = b$.

**Example 2.4.** Let $A = \begin{bmatrix} 1 & 1 \end{bmatrix}^T \in \mathbb{R}^{1\times 2}$ and let $b = \begin{bmatrix} 1 \end{bmatrix} \in \mathbb{R}^1$. Obviously $x_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}$ is a solution to $Ax = b$ of minimal support (with weight 1) while $x_2 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}$ solves $Ax = b$ with weight 2. However, $\|x_2\|_2 = \frac{\sqrt{2}}{2} < 1 = \|x_1\|_2$, indicating that the $\ell_2$-minimizer must be of smaller Euclidean length than the minimal support solution. Hence, minimizing $\ell_2$ distance may not recover a solution of minimal support.

Analytically, minimal $\ell_2$ problems generally appear similar to Example 2.4: minimal $\ell_2$ solutions tend to not be sparse, but instead have many nonzero elements small in magnitude.

In fact, there are many cases where the best $\ell_p$ approximation fails to even recover solutions of weight 1 for every $p > 1$.

**Proposition 2.5.** Suppose $A \in \mathbb{F}^{m\times(m+1)}$ and $b \in \mathbb{F}^m$ form a linear system $Ax = b$ as given in Definition 1.1. Assume $m > 1$.

Without loss of generality, let $G = RREF[A|b] = [I_{m\times m} : A'|b]$, and assume $|A'_i| = |G_{i,n-1}| = b_i = \left(\frac{1}{m}\right)$. Then a minimal support solution for the corresponding system $Ax = b$ must be greater in norm than the minimal $\ell_p$ solution for all $p > 1$, implying the $\ell_p$-minimization process fails to report a minimal support solution of weight 1 for all $p > 1$ for these systems.

*Proof.* Let $S = \{1, \ldots, m\}$. We have that

$$\|A_S x_S\|_p = \|b\|_p = \|A'\|_p = \left(\sum_{i=1}^{m} |b_i|^p\right)^{\frac{1}{p}} = \left(\frac{1}{m}\right)^{\frac{p-1}{p}} < \sum_{i=1}^{m} |b_i| = \sum_{i=1}^{m} \left(\frac{1}{m}\right) = 1.$$

Any $\ell_p$-minimizer must, therefore, be bounded sharply by 1 in its respective norm. Since the unique minimal support solution $x = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \end{bmatrix}$ is such that $\|x\|_p = 1$ for all $p > 1$, the $\ell_p$ minimizer cannot be the minimal support solution. ////

In this section, we have seen that two direct methods - that of setting independent variables to zero and letting dependent variables equal the entries of $b$, and that of finding the solution with minimal $\ell_2$ measurement (and, in general, with minimal $\ell_p$, $p > 1$ measurement) - generally fail to obtain minimal support solutions, and so cannot be relied upon as effective methods for minimal support recovery.

## 2.2 Finiteness of the Set of Minimal Support Solutions

Observations 2.1 and 2.3 combine to form an interesting quality about minimal support solutions to any consistent underdetermined linear system over any field: the set of such solutions must be finite.

**Proposition 2.6.** Let $Ax = b$ be a consistent underdetermined linear system with $A \in \mathbb{F}^{m\times n}$ of full row rank. Assume $k \leq m$ is the cardinality of any solution of minimal support. Then the number of minimal support solutions to $Ax = b$ must be finite and bounded by $\binom{n}{k}$.

*Proof.* Suppose $x$ is a minimal support solution of size $k$. We have that $Ax = A_S x_S = b$, where $S$ is the support of $x$ and $|S| = k$ and where $x_S \in \mathbb{F}^k$ is a vector with all nonzero entries corresponding to the nonzero entries of $x$ supported by $S$.

If a vector $y_S$ distinct from $x_S$ were such that $A_S y_S = b$, then by linearity, $A_S x_S = A_S y_S$ would imply $A_S(x_S - y_S) = 0$ which means $x_S - y_S$ is in the null space of $A_S$. Since $x_S$ and $y_S$ are distinct, there is a row $j < k$ of $x_S$ and $y_S$ where entries differ.

If $(y_S)_j = 0$, then creating the vector $y$ equalling $y_S$ on the indices of support $S$ and 0 otherwise would give $Ay = b$, meaning $\|y\|_0 < k$ is smaller weight than the minimal support weight for the system $Ax = b$, a contradiction. Additionally, if $(y_S)_j \neq 0$, $(x_S)_j - (y_S)_j = c$ for some nonzero element in the field. Therefore, $(c)^{-1}(-(x_S)_j)(x_S - y_S)$ is in the null space of $A_S$, and we would have that $z_S := x_S + (c)^{-1}(-(x_S)_j)(x_S - y_S)$ is such that $A_S z_S = b$, and constructing $z$ to be $z_S$ on the indices of support $S$ and 0 on the remaining $n - |S|$ rows would get $Az = b$. But row $s_j$ of $z$ would equal row $j$ of $z_S$, and this quantity is

$$z_{s_j} = (z_S)_j = (x_S)_j + -(x_S)_j c^{-1} c = 0,$$

again contradicting the assumption that the minimal support weight is $k$.

Hence, $A_S$ is injective, so that a minimal support solution to $Ax = b$ on a particular set of indices $S$ of size $k$ is unique. Thus $\arg \min_{x: Ax=b} (x) \leq \binom{n}{k} < \infty$. ////

## 2.3 Basis Pursuit: The $\ell_1$-Minimization Approximation for Minimal Support Solutions

In this section, we explore a method for recovering minimal support solutions to underdetermined systems of linear equations over the real field that shows more promise than other attempts at general approximation methods in Section 2.1.This is the method of *Basis Pursuit*, or of finding a vector in the solution set $x' + N$ of minimal $\ell_1$ norm. We call this element an "$\ell_1$ *minimizer*," and use the term similarly with $\ell_p$, $0 \leq p < \infty$.

An example that the method fails in some cases, and then example demonstrating the general underlying success of this method, are presented. Following this opening discussion, a brief introduction to *linear programming* is given, with the goal of demonstrating that a minimal evaluation for a linear *objective function* representing a case of Problem 1.2 is possible. Deeper analysis of the Basis Pursuit method is presented in Chapter 4.

### 2.3.1 Brief Discussion of Basis Pursuit

The following proposition allows use of the Simplex Method as a reliable algorithm for $\ell_1$-minimization. In terms of Chapter 3, this means $\ell_1$ minimization may be assessed in polynomial time with some adjustment to the standard usage of the Simplex Method here (see, for example, Kelner's development in (22)). Here, $x^+$ records the nonnegative entries of $x$ with zeros in the negative entries, and $x^-$ records the absolute value of the nonpositive entries of $x$ with zeros in the positive entries. That is, $x = x^+ - x^-$.

**Proposition 2.7.** For a given underdetermined system $Ax = b$ over the real field, finding

$$\min\left(\sum_{k=1}^{n}(x_k^+ + x_k^-)\right)$$

for the system

$$\begin{bmatrix} A & -A \end{bmatrix} \begin{bmatrix} x^+ & x^- \end{bmatrix} = b$$

with $x^+, x^- \geq 0$ is equivalent to finding $\beta = \min\limits_{x:Ax=b}(\|x\|_1)$ for the system $Ax = b$.

*Proof.* It is immediate that

$$\begin{aligned} Ax &= A(x^+ - x^-) \\ &= A(x^+) - A(x^-) \\ &= \begin{bmatrix} A & -A \end{bmatrix} \begin{bmatrix} x^+ & x^- \end{bmatrix} \end{aligned}$$

////

### 2.3.2 Failure of Basis Pursuit

The first example of this subsection is a counterexample that demonstrates that the Basis Pursuit approximation to Problem 1.2 does not always yield a minimal support solution.

**Example 2.8.** Let $A = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{4} \\ 0 & 1 & 0 & \frac{1}{4} \\ 0 & 0 & 1 & \frac{1}{4} \end{bmatrix}$ and let $b = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$. It is immediate that $x_1 = \begin{bmatrix} 0 & 0 & 0 & 4 \end{bmatrix}$ is the unique minimum support solution; note, however, that $\|x_1\|_1 = 3|0| + |4| = 4$ while the solution vector $\begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}$ is such that $\|x_2\|_1 = 3$. Hence, the minimal support solution is not a minimal $\ell_1$ solution to this system, which is at most measurement 3.

### 2.3.3 Linear Programming

To understand the approaches in this section, a brief introduction to *linear optimization* is necessary. Most of this material is paraphrased from (6) and (4). The proofs of many of these statements are tangential to the purpose of this section and are available for reference in these works.

A *standard linear program* or *optimization problem* is the problem of maximizing the real-valued *objective function* $f(x) = c \cdot x$ with $x, c \in \mathbb{R}^n$, subject to the *constraints*

$$Ax = b$$

$$x \geq 0$$

where $A \in \mathbb{F}^{m \times n}, m < n$ is of full row rank $m$ and where the equality and inequality constraints are considered elementwise.

One standard method for finding the maximum value for a linear program, the *Simplex Method*, is described informally in two basic steps: 1. Select a vertex in the *feasible* *region* at which the objective function is to be evaluated; 2. move from the selected vertex to an adjacent vertex in such a way that the objective function increases the most, and then decide whether another vertex selection further increases the function, or else decide that the optimal solution is found.

If one wishes to minimize instead of maximize, as we do below, the Simplex Method works with the obvious adjustments to the objective function and to the method of selecting vertices. This is accomplished typically via maximizing the *dual problem*, but by the simplicity of our selected example, we work directly with the original augmented matrix below.

### 2.3.4 The Success of Basis Pursuit

With a background on the simplex algorithm in hand, an example demonstrating the success of Basis Pursuit follows.

**Example 2.9.** As in Example 2.4 above, let $A = \begin{bmatrix} 1 & 1 \end{bmatrix} \in \mathbb{R}^{1\times 2}$ and let $b = \begin{bmatrix} 1 \end{bmatrix} \in \mathbb{R}^1$ as before. Let $x' = \begin{bmatrix} x^+ & x^- \end{bmatrix}$ with $x^+, x^- \geq 0$ such that $x = x^+ - x^-$. Recall that $\begin{bmatrix} A & -A \end{bmatrix} \begin{bmatrix} x^+ & x^- \end{bmatrix} = b = 1$ as in Proposition 2.7. Let $y = \sum\limits_{k=1}^{4} x'_k = \|x\|_1$. Our goal is to *minimize* $y$. We obtain the simplex augmented matrix:

$$\begin{bmatrix} 1 & 1 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 0 \end{bmatrix} \tag{2.1}$$

The six columns of this matrix are coefficients for $x'_1, \ldots, x'_4$, $y$, and $b$ respectively. The first row represents the system $\begin{bmatrix} A & -A \end{bmatrix} \begin{bmatrix} x^+ & x^- \end{bmatrix} = b = 1$ and the last row represents $-\sum\limits_{k=1}^{4} x'_k + y = 0$.

That all the coefficients for the $x'_k$ are equal to $-1$ in the bottom row indicates potential for pivoting at least one of $1, c$, $c = 1, 2, 3, 4$, in hopes that these row operations dispose of as many variables as possible. Since $x'_1 + x'_2 - x'_3 - x'_4 = 1$ and the nonnegativity of these variables imply that the sum of all four must be greater than or equal to one, $y$'s minimum value (as the sum of all four variables) *is* one. We now set this intuition in stone by taking the next step for the augmented matrix.

The minimal value for $y$ is attained by using row operations to exchange the candidate row for $e_1$, and setting the other three (necessarily free) variables to zero. Looking at Column 1 and Column 2 of row one, the (positive) ratio 1 is obtained from dividing their entries into $b = 1$. The last two columns cannot be pivots, for the results representing values for these variables are negative, contradicting the condition that these elements are nonnegative. Furthermore, adding Row 1 to Row 2 rids representation of both $x'_1$ and $x'_2$; $x'_3$ and $x'_4$ remain free with choosing either $x'_1$ or $x'_2$ as the pivot, and in either of these two cases, a 1 is dropped in the last column of row 2, and the final matrix is

$$\begin{bmatrix} 1 & 1 & -1 & -1 & 0 & 1 \\ 0 & 0 & -2 & -2 & 1 & 1 \end{bmatrix}.$$

Either choosing $x'_1 = 1$ or $x'_2 = 1$ as the pivot leaves us with $y = 1 + 2x'_3 + 2x'_4$, and since the latter two are free in either choice of pivot for $x'_1$ or $x'_2$ (along with the unchosen element for the possible pivot), we minimize $y$ by setting both to zero, obtaining the minimum possible value $y = 1$ at both the coordinates $x'_1 = x_1 = 1, x'_2 = x_2 = 0$ and $x'_1 = x_1 = 0, x'_2 = x_2 = 1$. We note that both of these are the minimal support solutions to the original equation $x_1 + x_2 = 1$, and as the Simplex Method obtains at least one of these two solutions by its design, this demonstrates that Basis Pursuit is able to successfully recover a minimal support solution by using the Simplex Method as an algorithm to recover a minimal $\ell_1$ solution.

## 2.4 Minimization of $\ell_q$, $0 < q < 1$

We conclude this chapter with a more advanced idea for approximating minimal support solutions via finding the minimizer of a system for $\ell_q$ "norms," defined as

$\left(\sum_{k=1}^{n} |x_n|^q\right)^q$, $0 < q < 1$ (note the difference of notation from $\ell_p$, $p > 0$). First, we establish a simple result linking $\ell_q$, $q > 0$, with $\ell_0$ which demonstrates that the number of nonzero entries inevitably approach one while zero entries stay zero in any sequence of decreasing numbers $q_j \to 0^+$.

**Proposition 2.10.** $\lim\limits_{q\to 0^+} \|x\|_q^q = \|x\|_0$ for any $x \in \mathbb{R}^n$.

*Proof.* Let $x_j$ and $x_k$ be nonzero entries of $x$ such that $x_j$ and $x_i$ are, respectively, the smallest and largest entries of $x$ in magnitude. Then $|x_j|^q \leq |x_i|^q$ and $|x_j|^q \leq |x_k|^q \leq |x_i|^q$ by concavity. We have $t|x_j|^q \leq \|x\|_q^q \leq t|x_k|^q$ for all $q > 0$, thus $\lim\limits_{q\to 0^+} |x_j|^q = \lim\limits_{q\to 0^+} |x_k|^q = 1$. Hence,

$$t = \|x\|_0 \leq \lim_{q\to 0^+} \|x\|_q^q \leq t = \|x\|_0.$$

////

Proposition 2.10 prompts an intuitive idea - since $\|x\|_q^q \to \|x\|_0$, perhaps the set of minimal support solutions $\arg \min\limits_{x:Ax=b} (\|x\|_0)$ can be approximated by examining the sets of solutions

$$\arg \min_{Ax=b} \left(\|x\|_{q_i}^{q_i}\right) \tag{2.2}$$

for a finite, decreasing sequence $1 > q_1, \ldots, q_i, \ldots q_j > 0$.

Foucart (12) verifies this intuition. The proof, which involves techniques explored in Chapter 4, is omitted; the only part of the proof pertinent to the aim of this exploration is that some, but not all, matrices $A$ guarantee unique recovery for every $s$-sparse vector in the manner explained in the theorem below.

**Theorem 2.11.** *(12) If every $s$-sparse vector $x$ is the unique solution of 2.2 for the system with $b := Ax$, $1 \geq q_i > 0$, then every $s$-sparse vector $x$ is also the unique solution of 2.2 for $0 < q_j < q_i$ and $b := Ax$.*

This shows that the $\ell_q$ approximation for Problem 1.2 at least gets no worse than the minimizer for $\ell_{q'}$, $1 \geq q' > q$.

If the $\ell_q$ approximation can be performed by an algorithm computed in roughly the same number of steps required for efficient $\ell_1$ approximation algorithms, this provides a very efficient minimal support approximation method for systems fit for Theorem 2.11. However, the next chapter shows that approximating the minimal support solution through use of $\ell_q$ minimizers as in Theorem 2.11 represents a process that does not improve the efficiency of constructing a comprehensive $\ell_0$ minimizer directly.

## 2.5 Conclusion

This section explores counterexamples to approaching Problem 1.2 in a direct way. Minimizing $\ell_q$ for $0 < q \leq 1$ are explored thoroughly in the remaining chapters; in Chapter 3, the $\ell_q$ minimization process is discovered to be, in a well-defined sense, "just as hard" as solving the minimal support solution directly, and in Chapter 4, both the power and limitations of the Basis Pursuit method are investigated prior to the presentation of Algorithm 4.2.2.

# CHAPTER 3. NP-HARDNESS OF PROBLEM 1.2

The main goal of the first section of this chapter is to establish the membership of the central problem of this thesis in the complexity class *NP-Hard*. We provide rigorous foundation for the claim that any general algorithm that recovers all minimal-support solutions to any given underdetermined system of linear equations must run in a number of steps that is not bound by any polynomial function of the size of the input (assuming that $P \neq NP$).

Though an easy implication of Natarajan's proof of the NP-Hardness of a similar problem (19) implies NP-Hardness over the real and complex fields, and though citations for the general NP-Hardness over the binary field appear as early as 1979 in Garey and Johnson (5), this result has never been explicitly given at the level of generality in Theorem 3.6. The implication of this result for our task of constructing a thorough recovery procedure is discussed in context in both the current chapter and in Chapter 4.

The second section dismisses the efficacy of Theorem 2.11 by establishing the NP-Hardness of finding solutions of minimal $\ell_q$ "norm" for $0 < q < 1$, thereby closing off a potentially rewarding approximation route for Problem 1.2. A surprising connection to another path for demonstrating NP-Hardness of Problem 1.2 is revealed in the process of describing and proving this section's main result.

Finally, we present another approach proving the NP-Hardness of Problem 1.2 over $\mathbb{F}_2$ via reducing the (NP-Complete) *bounded predecessor existence problem* in additive automata theory to instances of finding minimal support solutions for underdetermined systems over the binary field. This result was achieved prior to discovering the former, more powerful result in the first section, but is included for the insight gained from this substantially different approach.

## 3.1 NP-Hardness of Problem 1.2 Over Any Field

### 3.1.1 Complexity

The purpose of this section is to provide a brief introduction to complexity theory, culminating in the NP-Hardness of Problem 1.2.

Information in this section is summarized from (4) and (5). The purpose is to clarify NP-hardness of a problem and how to prove this membership within the context of formal mathematics. Following convention in the literature, we do not adapt the formality presented within the notation here explicitly in proofs except where the context calls for such exactness; rather, the purpose of this section is to establish the existence of a rigorous mathematical foundation for the theory that follows.

#### 3.1.1.1 Algorithms and Tractability

An *algorithm* is a step-for-step procedure intent on solving a *problem* $\Pi$. An *instance* of $\Pi$ is a particular assignment of fixed values of *parameters* of $\Pi$, which are *unquantified* or *free variables* of $\Pi$.

An algorithm aims to produce a solution to $\Pi$ given any *instance* $I$. We assume the algorithm has a fixed *encoding scheme* that converts input into a *string* formed from symbols that are elements of some finite set, called an *alphabet*.

The *time complexity function* $f(n)$ is a function representing the operations needed for the algorithm to solve $\Pi$ for an instance of (arbitrary) length $n$. If $f(n)$ is such that $f(n) \leq c(p(n))$ for some positive real number $c$ as $n \to \infty$, we say $f(n) \in O(p(n))$ (big-O) and say that the algorithm is *tractable*, or is a *polynomial time algorithm.* If no such *bounding* polynomial exists, the algorithm is considered *intractable.*

Restating an optimization problem as a *recognition problem* is key to the application of complexity theory. A recognition problem is a problem $\Pi$ such that a supplied instance $I$ generates an answer of *yes* or *no.*

#### 3.1.1.2 The classes P, NP, NP-Complete, and NP-Hard

We now conclude with the foundational definitions of P, NP, NP-Complete, and NP-Hard, and a methodology of how to prove that a recognition problem belongs to the class NP-Hard.

$\Pi$ is in *class P* if there exists an algorithm solving $\Pi$ and an accompanying polynomial $p(x)$ such that every instance $I$ is solved in no more than $p(|I|)$ steps.

If, for a recognition problem for $\Pi$, an algorithm and a polynomial $q$ exist that verify or deny any (not necessarily deterministically) generated guess of a yes-answer $x$ for an arbitrary instance $I$ of $\Pi$ in no more than $q(|x|, |I|)$ steps, we say $\Pi$ is in NP.

A recognition problem $\Pi_1$ is (polynomially) *transformable* to the recognition problem $\Pi_2$ if, given a string $x_1$ of length $n_{x_1}$, a string $x_2$ is constructed in polynomial time (in terms of $(p(n_{x_1}))$ for some polynomial $p$), such that $x_1$ is a yes-instance of $\Pi_1$ if and only if $x_2$ is a yes-instance of $\Pi_2$.

An alternate term often used for a successful transformation is to state that $\Pi_1$ is *embedded* into $\Pi_2$. Thus, a (often hypothetically posited) successful program for $\Pi_2$, by virtue of the embedding of $\Pi_1$ and the polynomial steps needed to reverse the embedding back to the original instance of $\Pi_1$, solves *any* instance of $\Pi_1$ in roughly the same number of steps it takes to solve $\Pi_2$.

If a problem $\Pi$ is such that all other problems in NP embed into $\Pi$, then we say $\Pi$ is in the class *NP-Hard.* If the problem itself is in NP, we say the problem is *NP-Complete.* The class NP-Complete (and therefore NP-Hard) is not empty (4) due to the Cook-Levin theorem, which demonstrates the NP-Completeness of the SATISFIABILITY problem. Since transformability is transitive, all that is needed to demonstrate NP-Completeness

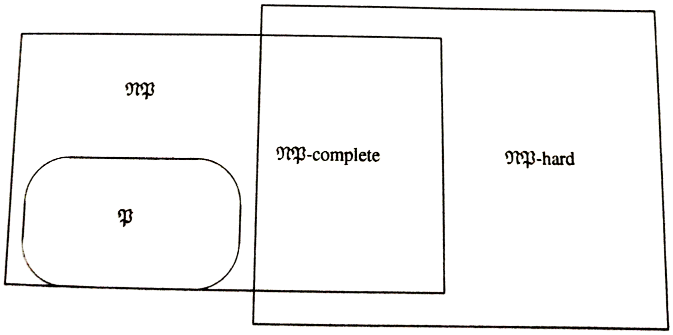

Figure 3.1 Diagram demonstrating relationships between various complexity classes (4).

is to show $\Pi \in$ NP and to show that any NP-Hard problem reduces (transforms) to $\Pi$. Figure 3.1 shows the relationship between these four classes of recognition problems.

In his seminal paper, Karp (15) establishes the membership in NP-Completeness of twenty-one well-known problems; due to the widespread knowledge and longevity of Karp's work, we begin the work in this paper with the assumption that the problems in (15) are NP-Complete.

### 3.1.2 NP-Completeness of 3-Dimensional Matching

The proof for the NP-Hardness of Problem 1.2 relies on the NP-Completeness of a problem known as 3-SET COVER, which in turn relies on the NP-Completeness of a problem known as 3-DIMENSIONAL MATCHING. In what follows, we provide the rigorous foundation required for the central result of this chapter.

The 3-DIMENSIONAL MATCHING (or 3D MATCHING) problem asks: given sets $U, V, W$ with the same cardinality, $T \subseteq U \times V \times W$, does there exist a subset $M \subseteq T$ of the same cardinality as $U$ such that whenever $(u_1, v_1, w_1), (u_2, v_2, w_2)$ are distinct triplets in $M$, $u_1 \neq u_2, v_1 \neq v_2, w_1 \neq w_2$?

$M$ is called a *perfect matching subset.* The existence of such a set relies on two distinct triplets in a potential candidate subset *not* matching in any coordinate; the name *matching* reflects the fact that the subgraphs representing the triplets in $M$ do not overlap in any way, and the term *perfect* comes from the inference that each element of the component sets belongs to exactly one coordinate of a triplet in $M$, a property shown below.

The proof of the NP-Completeness of 3-DIMENSIONAL MATCHING assumes the NP-Completeness of the SATISFIABILITY problem, one of Karp's 21 NP-Complete problems (4).

SATISFIABILITY asks whether a Boolean formula

$$\phi = \bigwedge_{k=1}^{m} \left( \bigvee_{i=1}^{n} \lambda_i^j \right)$$

where $\lambda_i^j$ are *literals* over $x_k, k \in [m]$ (that is, $\lambda_k$ is either $x_k$ or $\bar{x}_k$ in accordance with this variable's representation in clause $j$) is *satisfiable.*

$\phi$ is true if and only if at least one literal in every clause is true, an assignment called a *truth setting* for $\phi$.

As this problem is one of Karp's famous 21 NP-Complete problems, we assume without proof that this problem is NP-Complete (this is the famous Cook-Levin Theorem (4)).

For the SATISFIABILITY problem used below, we assume that every clause $C_j$ is composed only of Boolean variables (hence the clauses contain no True or False constants), and each clause has at least one Boolean variable; empty clauses are vacuously true by definition, and therefore are disregarded. Though each variable $x_i$ or $\bar{x}_i$ is examined in each clause $C_j$ by the construction below, neither need not appear in every particular $C_j$, as actual absence from a particular clause $C_k$ simply means the corresponding truth value has no effect on the truth value of $C_k$.

**Theorem 3.1.** *3-DIMENSIONAL MATCHING is NP-Complete.*

*Proof.* The proof is modified from (4), with notation adjustment and added detail. First, 3-D MATCHING is in NP, since a given candidate subset $S$ of $T$ is checked to verify that different triplets imply that all three such pairs of elements in the two triplets are pairwise different; if so, check whether $|S| = |U|$. This is done in roughly $\binom{|S|}{2}$ steps, a polynomial of the size of the input and the instance.

To show NP-Completeness, we embed the SATISFIABILITY problem. Take $\phi$ to be a Boolean formula over the alphabet $\{x_1, \dots, x_n\}$ with clauses $C_1, \dots, C_m$.

Construct the set

$$U = \{x_i^j, \bar{x}_i^j, i \in [n], j \in [m]\},$$

which contains one copy of each literal and its negation for each variable in the alphabet and each clause in $\phi$.

Then, construct

$$V = \{a_i^j, i \in [n], j \in [m]\} \cup \{v_j, j \in [m]\} \cup \{c_i^j, i \in [n-1], j \in [m]\}$$

and likewise construct

$$W = \{b_i^j, i \in [n], j \in [m]\} \cup \{w_j, j \in [m]\} \cup \{d_i^j, i \in [n-1], j \in [m]\}.$$

We have that $|U| = |V| = |W| = 2mn$. Hence any $M$ by assumption is such that $|M| = |U| = 2mn$.

Take

$$T = \{(a_i^j, b_i^j, x_i^j)\} \cup \{(a_i^{j+1}, b_i^j, \bar{x}_i^j)\} \cup \{(v_j, w_j, \lambda_j)\} \cup \{(c_i^j, d_i^j, \lambda_k)\} \subseteq V \times W \times U$$

where in the first triplet set in the union we have $i \in [n]$, $j \in [m]$, in the second triplet set we have $i \in [n], j \in [m]$ with $a_i^{m+1} = a_i^1$, in the third triplet set we have $j \in [m]$, and in the fourth triplet set we have $i \in [n-1], j \in [m], k \in [m]$. The first two sets in this union are together of cardinality $2mn$, and the last elements in every triplet in these two sets partition $U$.

The elements $\lambda_j$ and $\lambda_k$ in the last two triplet sets in $T$ are as follows: in the third set of the union, one of either the triplet $(v_j, w_j, x_i^j)$ or $(v_j, w_j, \bar{x}_i^j)$ is in the third set in accordance to which of $x_i^j$ or $\bar{x}_i^j$ is in the clause $C_j$, for each variable in the clause. In other words, the third set represents the *exact formula* of $\phi$, tying each variable in $C_j$ with its proper clause by attaching it to the corresponding $v_j, w_j$ triplet representing $C_j$. This means there are as many elements in this set as there are variables in all the clauses of $\phi$, which by assumption that the clauses are nonempty is at least $m$. For the fourth set, for *each* $\lambda_k \in U$ there are the triplets $(c_i^j, d_i^j, \lambda_k)$ for all $i \in [n-1]$ and for all $j \in [m]$. The fourth set is then the largest set, of size $m(n-1) * 2mn$, because each element in $U$ corresponds to exactly $m(n-1)$ elements in this fourth set.

Notice $T$ is determined by $\phi$.

For the first direction of the proof, assume that a perfect matching subset $M$ of $T$ exists. Remember that $M$ is a perfect matching set if and only if $|M| = |U| = 2mn$ and *no* element matches in any pair of distinct triplets selected from $M$.

To avoid coordinate equality and due to $|M| = |U|$, it follows that every $x_i^j$ and $\bar{x}_i^j$ are in exactly one triplet of $M$.

A maximum of $m(n-1)$ triplets from the fourth set are chosen without overlap of the $c, d$ elements, and a maximum of $m$ triplets from the third set are chosen without overlap from the $v, w$ elements. This means at least $mn$ triplets from the first two subsets are chosen for a perfect matching subset $M$ of $T$. Since selecting more than $mn$ triplets from Sets 1 and 2 equates $a$, $b$ elements for at least one distinct triplet in $M$, exactly $mn$ triplets must be taken from Sets 1 and 2, which means, in turn, that exactly $m$ triples are chosen from Set 3 and exactly $m(n-1)$ triples are chosen from Set 4, according to the above bounds and due to the requirement that $|M| = 2mn$.

From the first two sets, suppose $(a_i^{k+1}, b_i^k, \bar{x}_i^k)$ is chosen for some $i \in [n]$ and some $k \in [m]$; the element $a_i^{k+1}$ is equal to the first position in the triplet $(a_i^{k+1}, b_i^{k+1}, x_i^{k+1})$, so a perfect matching subset must select $(a_i^{k+2}, b_i^{k+1}, \bar{x}_i^{k+1})$ to avoid coordinate equality. This means $(a_i^{k+2}, b_i^{k+2}, x_i^{k+2})$ is not selected by $M$, which means $(a_i^{k+3}, b_i^{k+2}, \bar{x}_i^{k+2})$ is selected by $M$, and so forth. This shows that if $\bar{x}_i^j$ is selected in a triplet for any particular $i$ and $j$, then for this fixed $i$ and *every* $j \in [m]$, *only* the triplets with $\bar{x}_i^j$ are selected by

$M$, and since $mn$ coordinates from the first two subsets must be selected, *exactly all* $m$ triplets of this form are selected for this particular $i \in [n]$. A similar argument shows that if a triplet in $M$ has the element $x_i^j$, then for this $i$, *exactly* those triplets with $x_i^j$ for *all* $j \in [m]$ are selected for this particular $i$; furthermore, this selection is made for each $i \in [n]$ to attain the required $mn$ triplets from these first two sets.

The variable $x_i$ or $\bar{x}_i$ selected for $M$ from the $a_j, b_j$ triplets above cannot match the values $\lambda_j$ in the $m$ $(v_j, w_j, \lambda_j)$ coordinates that $M$ necessarily seletcts from the third subset of $T$. Since one triplet $(v_j, w_j, \lambda_j)$ is selected by $M$ for each of the $m$ clauses $C_j$ in $\phi$, setting $\lambda_j$ to true for each such triplet selected by $M$ provides a satisfiable setting for $\phi$, since the $\lambda_j$ are assumed to be variables in $C_j$ and all $m$ $C_j$ are represented in this way. No $\lambda_j$ overlap, as each $j$ for $\lambda_j = x_i^j$ or $\bar{x}_i^j$ is distinct even if $i$ is the same for distinct triplets from the third set. Hence if a perfect matching subset $M$ of the set of triplets $T$ determined by $\phi$ exists, $\phi$ is satisfiable.

Note that the remaining $m(n-1)$ selections from the fourth subset of $T$ are selected to fulfill the requirement for $M$ of selecting triplets with elements equal to the remaining $m(n-1)$ values from $U$ that $M$ has not yet selected. Due to the comprehensive representation of each element of $U$ in this fourth set, this selection is made with no individual element equalling any previously chosen element.

See Figure 3.3 for a visual demonstration of this process.

On the other hand, suppose $\phi$ is satisfiable.To form a perfect matching subset of the set of triplets $T$ constructed from $\phi$, select $m$ elements $(v_j, w_j, \lambda_j)$ from the third subset such that setting the variables $\lambda_1, \ldots, \lambda_m$ to true satisfies $\phi$. Select the remaining $mn$ $a, b$ triplets as follows: if $\lambda_j = x_i^j$, select all triplets $(a_i^{j+1}, b_i^j, \bar{x}_i^j)$ for all $j \in [m]$ and if $\lambda_j = \bar{x}_i^j$ select the triplets $(a_i^j, b_i^j, x_i^j)$ for all $j \in [m]$. Select the remaining required $m(n-1)$ triplets from the fourth set so that no matching occurs; as all variables in $U$ are available, this selection is made with no resulting matching elements. This demonstrates that if $\phi$ is satisfiable, a matching subset $M$ of the triplet set $T$ determined from $\phi$ above exists.

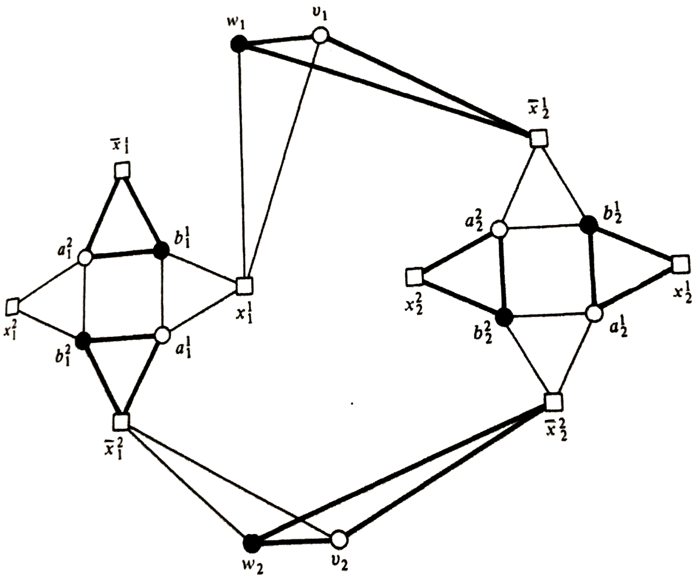

Figure 3.2 A generic 3-D matching representation of a Boolean formula (9).

Therefore, any program solving 3-D MATCHING solves the SATISFIABILITY problem. This embeds an NP-Complete problem, SATISFIABILITY, in 3D-MATCHING in polynomial time, and the first paragraph of the proof shows 3D-MATCHING is in NP. Hence 3D-MATCHING is NP-Complete. The proof is complete, as well. ////

### 3.1.3 Matrix and Vector Representation of Theorem 3.1

In this subsection, we reprove Theorem 3.1 in terms of a matrix representation for SATISFIABILITY. The goal to prove that finding a minimal support solution to a system of linear equations over any field is a problem in the class NP-Hard motivates this re-examination.

For a logical Boolean formula $\phi$, create an $m \times n$ matrix $A$ containing only the elements $-1, 0,$ and $1$ as follows: if clause $C_j$ has variable $x_i$, place $A_{j,i} = 1$; place a $-1$ in this position if $\bar{x}_i$ is represented in $C_j$ and place a 0 in $A_{j,i}$ if neither $x_i$ nor $\bar{x}_i$ appear in $C_j$.

The matrix $A$ represents $\phi$ in the sense that the rows correspond to the clauses of $\phi$ and the columns correspond to variables $x_1, \ldots, x_n$; an entry $1_{j,i}$ is present if and only if $x_i$ appears in $C_j$, an entry $-1_{j,i}$ is present if and only if $\bar{x}_i$ appears in $C_j$, and a zero entry is in position $j, i$ of $A$ if and only if neither $x_i$ nor $\bar{x}_i$ appears in $C_j$.

SATISFIABILITY is represented as asking whether there is an $n \times 1$ vector $z$ whose entries are 1 or $-1$, such that $Az = \mathbf{1}$, where $\mathbf{1}$ is the $m \times 1$ vector of all $1's$. The computation for $Az$ is done with the following arithmetic: row $j$ of $\mathbf{1}$, computed as $\sum\limits_{i=1}^{n} A_{j,i} z_i$ as normal, equals 1 if and only if at least one of $A_{j,i} z_i = 1$ for some $i$; otherwise, the sum is equal to $-1$. A sum of all zeros is defined to be 1, as this indicates that an empty clause is present in $C_j$, and therefore vacuously true, so the same assumption as above can allow us to disregard such rows.

This means that the equation $Az = \mathbf{1}$ is satisfied if and only if for all $j \in [m]$, at least one $i \in [n]$ is such that $A_{j,i} = z_i = 1$ or $A_{j,i} = z_i = -1$, as multiplication operates in the normal way (representing the logical opposite of the XOR operation with 1 and $-1$, and equalling 0 otherwise).

The problem of finding such a vector $z$ is therefore equivalent to the problem of examining whether a truth setting for $x_1, \ldots, x_n$ exists that make every clause $C_j$ true, meaning that $z$ represents a *truth setting* for the variables $x_1, \ldots, x_n$ (and hence a truth setting for the complements $\bar{x}_1, \ldots \bar{x}_n$). Again, by the Cook-Levin theorem, we know that SATISFIABILITY is NP-Complete.

The discussion above proves the following.

**Proposition 3.2.** Every logical formula $\phi$ is represented as a matrix $A$ in the way given above, and the SATISFIABILITY problem is solved, if and only if there is an $n \times 1$ vector $z$ such that $Az = \mathbf{1}$, where $\mathbf{1}$ is the $m \times 1$ vector with a 1 in each row, and where the computation $Az$ is performed using the arithmetic scheme defined above.

From this perspective, we prove that the 3D MATCHING problem is NP-Complete by embedding SATISFIABILITY to a special case of 3D MATCHING.

**Theorem 3.3.** *3D MATCHING is NP-Complete.*

*Proof.* By Proposition 3.2, each case of SATISFIABILITY is exactly represented as a problem of finding a solution to a linear system under the previously given arithmetic, where construction of a matrix $A$ corresponds to $\phi = \bigwedge_{k=1}^{m} \left( \bigvee_{i=1}^{n} \lambda_i^j \right)$ and construction of the problem of whether there exists an $n \times 1$ vector $z$ such that $Az = \mathbf{1}$ is in accordance with the previous discussion.

Embedding into 3D MATCHING then involves construction of the $2mn$-sized set $U$, which, in this representative case, contains each of $1_{j,i}$ and $-1_{j,i}$ for each entry $j, i$ of $A$; the set $V$ creates elements $a_{j,i}$ for each entry in $A$, $v_j$ for each row of $A$, and $c_{j,i}$ for each entry in the first $n-1$ columns of $A$; likewise, the set $W$ creates $b_{j,i}$ for each entry of $A$, $w_j$ for each row of $A$, and $d_{j,i}$ for each entry in the first $n-1$ columns of $A$.

The triplet set $T \subset U \times V \times W$ is represented, as above, in four sets:

$$T = \{(a_{j,i}, b_{j,i}, 1_{j,i})\} \cup \{(a_{j+1,i}, b_{j,i}, -1_{j,i})\} \cup \{(v_j, w_j, \lambda_j)\} \cup \{(c_{j,i}, d_{j,i}, \lambda_k)\}$$

where $i \in [n]$ and $j \in [m]$ for the first two sets, $j \in [m]$ in set three, and, for the fourth set in the union for $T$, $i \in [n-1]$, $j \in [m]$ and $k \in [m]$.

As before, the $\lambda_j$ are representative of the exact literals present in the clause $C_j$; that is, the third set contains $(v_j, w_j, 1_{j,i})$ if and only if the element $A_{j,i} = 1$, and $(v_j, w_j, -1_{j,i})$ if and only if $A_{j,i} = -1$, and no other elements are constructed for Set 3. Finally, the fourth set contains, for each element $1_{k,l}, -1_{k,l} \in U$, the $(2mn)(m)(n-1)$ triplets $(1_{k,l}, c_{j,i}, d_{j,i})$ and $(-1_{k,l}, c_{j,i}, d_{j,i})$ for every $j \in [m]$ and $i \in [n-1]$.

The $m \times n$ matrix $A$ representing the clause $\phi$ is sent and this construction for $T$ is made, noting that the construction is made in $O(m^2 n^2)$ steps (based on the largest constructed set of $T$, which is the fourth set in the union), which is a number of steps that is a polynomial function of the size of the input.

If a perfect matching subset $M$ of $T$ is assumed to exist, the logic follows identically with the above proof in Theorem 3.1; namely, exactly $m$ elements of the third set of $T$

are selected, representing one entry of $1_{j,i}$ or $-1_{j,i}$ for some $i \in [n]$ and for each $j \in [m]$ row of $A$. For the matrix representation of this problem, this direction instructs us to set up $z$ such that $z_i = 1$ if and only if $A_{j,i} = 1$ and $z_i = -1$ if and only if $A_{j,i} = -1$. If any $i$ is shared between different rows $j$ and $k$, it cannot be the case that $A_{j,i} \neq A_{k,i}$ (ignoring index); if so, one of these two triplets in $M$ has a coordinate equal to a coordinate from one of the $mn$ $a, b$ triplets selected from the first two sets of $T$ as in the proof for Theorem 3.1. This shows the constructed vector $z$ is well-defined.

The rest of the values selected for $z$ are not relevant; $\sum\limits_{i=1}^{n} A_{j,i} z_i = 1$ for each row $j$ due to each row having one entry in some column $i$, $A_{j,i}$, whose nonzero value is exactly the same value in $z_i$, meaning $A_{j,i} z_i = 1$ for at least one $i$ in each of the $m$ rows of $A$, so that $Az = \mathbf{1}$ as required.

Assuming we are given a matrix $A$ representing $\phi$ and a vector $z$ such that $Az = \mathbf{1}$, for each row of $A$, a column exists with index $i$ whose corresponding entry of $A$, $A_{j,i}$, is equal to $z_i$. Selecting this set (considering difference of index $j$) of size $m$ for $M$, selecting mn $a, b$ triplets such that none of the selection has an entry equal to the values for the $A_{j,i}$ chosen from the third set of triplets of $T$, and selecting $m(n-1)$ triplets from the fourth set of $T$ whose entries from $U$ do not equal the values for the chosen $A_{j,i}$ from the third set of $T$ or the $mn$ $U$-values chosen from the first two sets of $T$ (and making sure to choose a different $c, d$ triplet for each choice, which is possible because we make exactly m(n-1) choices in accordance with the above description) demonstrates the presence of a perfect matching set $M \subset T$ of size $|M| = 2mn$, as required. SATISFIABILITY is therefore embedded into 3D-MATCHING, showing that 3D-MATCHING is in the class NP-Complete. ////

### 3.1.4 NP Completeness of 3-SET-COVER

Theorem 3.3 directly leads to the NP-Completeness of 3-SET-COVER. The proof is not given full form in (4); the details not included with this source are filled in below.

**Theorem 3.4.** *The following problem, 3-SET COVER, is NP-Complete: given a collection* $\mathbf{C} = \{C_i, i \in [N]\}$ *of* 3*-element subsets of* $[m]$*, does there exist a subcollection* $C_j, j \in J$ *of* $[m]$*,* $J \subset [N]$ *of* $\mathbf{C}$*,* $C_i \cap C_j = \varnothing$ *for all* $i \neq j$*, such that* $\bigcup_{j \in J} C_j = [m]$ *with* $C_i \cap C_j = \varnothing$*?*

*Proof.* 3-SET COVER is in NP; given any subcollection $\{C_j\}$ of $\mathbf{C}$, check if each element of $[m]$ belongs to exactly one of the subsets in the subcollection. This both verifies that the subcollection is nonoverlapping and covers $[m]$. This is done in $\binom{|C_j|}{2}$ steps, and so is polynomial to the size of the input and the guessed solution.

3-SET COVER embeds 3-DIMENSIONAL MATCHING; set $S = U \cup V \cup W$. Since 3-D MATCHING assumes $|U| = |V| = |W|$, $|S| = 3k$ for some $k \in \mathbb{N}$.

First, index each element of $S$. Let $C$ be the collection of subsets of $|S|$ of size three, with each subset containing the three indices of each triplet in $T$.

If $\mathbf{C}$ contains a subcollection that partitions $|S|$, then the subset of $T$ whose triplets are each indexed by a set in this subcollection do not have equal entries for any distinct pair of triplets, because the corresponding subcollection of $\mathbf{C}$ of their indices does not overlap. Furthermore, since the subcollection covers $|S|$, then the corresponding triplets in $T$ are of size $\frac{|S|}{3} = |U|$, meaning the subcollection of $T$ with indices denoted by the subcollection of $\mathbf{C}$ that partitions $|M|$ is a perfect matching subset of $T$.

Likewise if $T$ contains a perfect matching subset $M$, the indices of the elements in each triplet of $M$ are distinct from each other triplet in $M$, and since $|M| = |U|$, this means there are $3|M| = 3|U| = |S|$ nonoverlapping subsets of such indices. Hence the subcollection of $C$ whose subsets correspond to the indices of the triplets of $M$ forms a partition of $|S|$.

3D-MATCHING is therefore embedded in 3-SET-COVER, and together with the opening argument we conclude that 3-SET-COVER is in the class NP-Complete.

////

### 3.1.5 NP-Hardness of Problem 1.2

The proof to Theorem 3.6 (and, in the next section, the proof to Theorem 3.9) uses a *special case* of Problem 1.2, the latter of which we deem MINIMAL SUPPORT in the tradition of capitalizing problems analyzed in the discipline of combinatorial optimization. A *special case* of a problem is a set of instances of the problem that must adhere to a set of *constraints*. The special case of MINIMAL SUPPORT employed in this chapter asks whether the minimal solution is of weight bounded by $cm$, where $c$ is such that $c \in (0, 1]$ and $cm \in \mathbb{N}$. Theorem 3.9 also relies on constraining the maximum weight for a minimal support solution, with the constraint also a linear function of the input. With the NP-Completeness of Theorem 3.4 in hand, we now show that Problem 1.2 is NP-Hard over any field.

**Lemma 3.5.** Given a matrix $A \in \mathbb{F}^{m\times n}$ of full rank and a vector $b \in \mathbb{F}^m$, $m < n$, the problem $c, d$-MINIMAL SUPPORT of finding a minimal support solution to the system $Ax = b$ with weight bounded by the natural number $cm - d$, $0 < c \leq 1$, is a special case of MINIMAL SUPPORT, so that showing $c, d$-MINIMAL SUPPORT is NP-Hard in turn implies MINIMAL SUPPORT is NP-Hard.

Further discussion about special cases can be found in (4). It is obvious that the NP-Hardness of a special case of a problem implies the NP-Hardness of the problem in general; the process is known as a proof of NP-Hardness *by restriction.*

**Theorem 3.6.** *Problem 1.2, called MINIMAL SUPPORT, is NP-Hard over any field* $\mathbb{F}$.

*Proof.* We embed 3-SET-COVER into the special case problem $\frac{m}{3}$-MINIMAL SUPPORT of MINIMAL SUPPORT, extending the related result over the real and complex fields given in (19).

Suppose we are given a collection $\{C_j, j \in [N]\}$ of 3-element subsets of $[m]$. Define a matrix $A \in \mathbb{F}^{m\times N}$ by $A_{i,j} = 1$ iff $i \in C_j$ and 0 iff $i \notin C_j$. Let $b = \mathbf{1}$ be the vector in $\mathbb{F}^m$ with each element equal to 1. Since $N < \binom{m}{3}$, construction of this matrix is performed in a number of steps equal to a polynomial function of the size of the instance of 3-SET-COVER.

Construct $A$ and $b = \mathbf{1}$ as above. Let $A_S$ be the submatrix of $A$ whose only columns are those columns of $A$ indexed exactly by $S$, the indices of support for $x$; again, let $x_S \in \mathbb{F}^{|S|}$ be equal to the values of the vector $x$ only on its support $S$. We have that a minimal support solution $x \in \{0,1\}^N$ to the system $Ax = \mathbf{1}$ exists and is of weight bounded by $\frac{m}{3}$ if and only if the matrix $A_S$ is such that each row has exactly one 1 with the rest of the elements zero. The latter condition occurs iff each column of $A$ has exactly three ones as row entries and exactly $(m-3)$ row entries with zeros; also, note $\|b\|_0 = m = 3 * \frac{m}{3} = \|A_S x_S\|_0$. $A$ is in this form if and only if the subcollection of $\{C_j\}$ indexed by $S$ represents a partition that covers $[m]$, since the preceding description implies that $A_S$ has no overlapping ones in any row together with the fact that $A_S x_S = Ax = \mathbf{1}$.

Therefore, $\frac{m}{3}$-MINIMAL SUPPORT affirms if and only if 3-SET-COVER affirms given the above construction. 3-SET-COVER is therefore embedded into $\frac{m}{3}$-MINIMAL SUPPORT. As $\mathbb{F}$ is arbitrary, this imbeds the NP-Complete 3-SET-COVER problem into MINIMAL SUPPORT over any field, meaning MINIMAL SUPPORT is NP-Hard over every field, by Lemma 3.5.

////

For any designer of an algorithm purposed for comprehensive and unconstrained minimal support recovery for any underdetermined, consistent linear system over any field, the message sent by Theorem 3.6 is a greeting card taped to a sledgehammer. The consequence from this theorem is that such an algorithm must generally require a number of steps not bounded by any polynomial function of the size of the instance, and no program that recovers minimal support solutions in a number of steps bounded by a polynomial function of the size of the input is guaranteed success, prompting the two technically trivial but pragmatically vital corollaries below.

**Corollary 3.7.** Any algorithm designed for full minimal support recovery for any consistent, underdetermined system of linear equations over any field must be intractable.

**Corollary 3.8.** Any average-case polynomial time algorithm solving or approximating Problem 1.2 must fail to produce minimal support solutions for some set of consistent underdetermined linear systems over any field for which the algorithm is applicable.

## 3.2 NP-Hardness of Minimizing $\|x\|_q^q$ for $0 < q < 1$

In this small section, we revisit the hopeful result established by the last section of Chapter 2, a result hinted toward by Theorem 2.11. We also discover a surprising tie between this problem and Problem 1.2 in the unexpected similar applicability of the arguments for Theorem 3.9 to Problem 1.2 itself.

The minimization method $\arg\min\limits_{x:Ax=b}(\|x\|_q^q)$, $0 < q < 1$, is introduced to the class NP-Hard via reduction from the *partition problem,* which asks if a multiset of integers $S$ has a partition of into two nonoverlapping subsets of equal sum. This problem is represented in matrix form below, following a suggestion from Foucart (12) that this route should establish the required NP-Hardness result.

Surprisingly, following these same arguments establishes NP-Hardness of Problem 1.2 over fields of characteristic zero; as the proof is nearly identical, it is presented in parallel with the main result here.

As the partition problem is one of Karp's original 21 NP-Complete problems (15), we assume its NP-Completeness from the outset. Similar to Theorem 3.6, Theorem 3.9 relies on a special case of both the $\ell_q$ minimization and $\ell_0$ minimization problems, specifically the constrained case where the measurement of the respective minimizer is bounded by $n$ in both cases, where $n+1$ is the number of rows in the $(n+1 \times (2n)$-matrix constructed in the proof that follows.

**Theorem 3.9.** *The optimization problem* $\arg\min\limits_{x:Ax=b}(\|x\|_q^q)$ *for any* $0 < q < 1$*, and the optimization problem* $\arg\min\limits_{x:Ax=b}(\|x\|_0)$ *over the real and complex fields, are both in the class NP-Hard.*

*Proof.* The proof relies on reduction of the partition problem, which asks whether a multiset of nonzero integers $S$ has a partition of two non-overlapping subsets with equal sum.

First, label the elements of $S$ as $a_1, \dots, a_n$ and construct the following $(n+1) \times (2n)$ matrix $A$ :

$$A = \left[\begin{array}{c|c} a_1, \dots, a_n & -a_1, \dots, -a_n \\ \hline I_{n\times n} & I_{n\times n} \end{array}\right]$$

and let $b$ be the vector with $b_1 = 0$ and $b_i = 1$ for $i = 2, \dots, n+1$.

The partition problem is equivalent to searching for a vector $x \in \{0,1\}^{2n}$ of weight $n$, with 1 in exactly one of the two rows of $x$ whose index corresponds to the index of exactly one of the two columns of $A$ with $a_k$ or $-a_k$ as the first row entries; 0 is placed in the other index. In other words, $x$ has exactly one 1 and exactly one 0 in rows $k \leq n$ and $n+k$, respectively, for each $1 \leq k \leq n$ and additionally solves $Ax = b$ if and only if the supported entries correspond to the indices of the elements forming a solution to partition ($-1, 0$ pairs assigned above work as well; we ignore this case and proceed assuming $x \geq 0$ WLOG as n nonzeros are still placed regardless of whether 1 or $-1$ is used).

It remains to show that such a solution must be a unique solution of minimal size with respect to the $\|x\|_q^q$ metric in such "yes" cases of partition. Additionally, we will show that "no" cases necessitate a larger weight minimal support solution than "yes" cases, so we follow Lemma 3.5 in constraining the size of the minimizer. We do so likewise for the parallel proof for $\ell_0$ minimization. Specifically, for both problems, we embed into the special case where the bound on the minimal support is $n = (n+1) - 1$. The proof of these statements are as follows.

For any other solution $x$ to $Ax = b$ different from the form given in the second paragraph, it must be the case that at least one pair of entries $x_k$ and $x_{k+n}$, $k \leq n$, are both nonzero in order to achieve a 1 in row $k$ of $b$. Furthermore, since

$$(A_k x_k + A_{k+n} x_{k+n})_1 = a_k x_k - a_k x_{k+n} = a_k(x_k - x_{k+n}) = 0,$$

and as $a_k$ is assumed to be nonzero, it follows that $x_k = x_{k+n}$, and therefore that $2x_k = 2x_{k+n} = 1$, revealing $x_k = x_{k+n} = \frac{1}{2}$. The rest of $x$ must have 1 and 0 placed as described above in accordance with elements in the first row that are equal to a sum of other elements, or else must have more pairs $x_k$ and $x_{n+k}$ both $\frac{1}{2}$, to ensure that $b$ has 1 in rows $2, \ldots, m$ of $b$ and 0 in row 1.

Since $\left(\frac{1}{2}\right)^q > \frac{1}{2}$, $|x_k|^q + |x_{k+n}|^q > x_k + x_{k+n} = 1$, and it follows that any minimal $\|x\|_q^q$ solution $x$ to $Ax = b$ must be such that $\|x\|_q^q \geq n$, with equality if and only if $x$ is a vector solving partition as described in the second paragraph. Since two nonzero elements in $x_k$ and $x_{k+n}$ only increase the weight of the solution representing a solution to partition as described above, the minimal support solution $x$ is such that $\|x\|_0 \geq n$, with equality if and only if $x$ is in fact the vector representing the solution to partition.

Hence, for both the $\ell_q$ and $\ell_0$-minimization problems, the partition problem is affirmed if and only if the vector $x$ of the respective minimal measurement is exactly of $\ell_q$ (respectively, $\ell_0$) weight $n$, which occurs iff $n$-MINIMAL SUPPORT and $n$-$\ell_q$-MINIMIZER affirm respectively. This embeds the NP-Complete partition into a special case for both problems, meaning the $\ell_q$ minimization problem of finding $\arg\min\limits_{x:Ax=b}(\|x\|_q^q)$ and the $\ell_0$ minimization problem in Problem 1.2 are both NP-Hard over the real and complex fields by Lemma 3.5.

////

This section dispenses of the potentially fruitful $\ell_q$-minimization approach described by Theorem 2.11; as a result, the problem is just as hard as Problem 1.2. In the process of completing the suggested proof from (12), the additional surprising application to

Problem 1.2 itself is noted and formally demonstrated along with the main intended result. This unexpected treasure provides valuable insight to the presence of an intimate tie between the fate of the $\ell_q$ minimization process of Theorem 2.11 and the fate of Problem 1.2 itself.

## 3.3 Complexity Revisited: NP-Hardness of Problem 1.2 over the Binary Field

As seen above, novel constructions in combinatorial optimization can lead to different proofs of problems whose complexity class membership has already been shown, providing interesting connections that are not assessed from the original proof. This section likewise reflects this same intention; we repeat the proof for Theorem 3.6 over the finite field $\mathbb{F}_2$ through a different proof than the ones illustrated in 3.6 or 3.9.

We reapproach Problem 1.2 over $\mathbb{F}_2$ through exploring its connection with the Bounded Predecessor Existence Problem (BPEP) from automata theory. This methodology demonstrates an important application of minimal support solution recovery for underdetermined systems via analysis of an incidence matrix for a graph; finite field matrix-vector multiplication with the incidence matrix is shown to exactly represent computation of immediate successor states of a finite-state automata attending to the graph, and minimal support solution recovery is employed to uncover a minimal-weight immediate predecessor state to a given maximal-weight state presented for the graph by the automata.

We begin with elementary notions of graph theory and basic building blocks that set the stage for finite-state automata, and then proceed to define automata and the Bounded Predecessor Existence Problem (BPEP) for an automata assigning values from $\mathbb{F}_2$ to nodes of an undirected graph. The chapter concludes with the proof of NP-Hardness of Problem 1.2 over $\mathbb{F}_2$ by embedding the NP-Complete BPEP as a special case.

### 3.3.1 Additive Automata on Graphs and Predecessor Configurations

Much of what follows is owed to (9), (2), or (4), and any unfamiliar terms may be referenced in these works.

An *automata* is defined to be a free monoid $A^*$ together with a set of *states* $S$ where each element of the monoid represents a rearrangement or *permutation* of lists of size $|S|$, and for each $a, b \in A^*$ and for the identity $e$ of $A^*$ we have that $(a * b) \circ s = a * (b \circ s)$ and $e \circ s = s$ for all $s \in S$; here, $a * b$ represents the monoid operation and $\circ$ represents the action of the monoid on elements of $S$.

If $G$ is an undirected, finite graph with nonisolated vertices $V = \{v_1, \ldots, v_n\}$ and edges $E \in V \times V$, a *cellular automata on* $G$ is an automata with Abelian monoid $\mathbb{F}$ and state set $S$ together with a collection of *configurations* $X : V \to S$ and a *local rule* $\rho : \{X_v\} \to S$ for all $v \in V$. Here, $X_v$ is a *local configuration* $X_v : \Gamma(v) \to \mathbb{F}$ defined by $X_v(u) = X(u)$ for each $u$ adjacent to $v$. The action of $\rho$ on each local configuration $X_v$ of a vertex $v$ of $G$ is defined as $\rho(X_v) := \sum\limits_{u \in \Gamma(v)} X(u)$.

The symbol $\rho^+$ denotes a local rule using the closed neighborhood $\Gamma^+(v)$.

The *global rule* for the automata, $\rho(X)$, is defined to be $\rho(X)(v) = \rho(X_v)$ for all $v \in V$. We call $Y = \rho(X)$ the immediate *successor* state of $X$, and $X$ is referred to as an immediate *predecessor* state of $Y$. In a slight abuse of notation, immediate successors and immediate predecessors may be referred to simply as successors and predecessors when the context is clear on the intent of use.

The adjacency matrix $A$ of $G$ in this case has elements 0 or 1 with arithmetic considered under the rules of the base field $\mathbb{F}$.

A configuration $X$ is represented as a column vector over $\mathbb{F}$; we immediately have that $Y = \rho(X) = AX$ through standard matrix multiplication over $\mathbb{F}$.

The automata used in this work are based on undirected graphs. Successor configurations for each vertex (or node) of an automata on an undirected graph are considered to be computed simultaneously. This is opposed to automata on directed graphs, where successor states for vertices are computed in order of a directed walk through the vertices of $G$; in the latter case, successor states may depend on previously computed successor states for neighbors reached first by the updating configuration, but for undirected graphs, each successor state is computed from the value present in each neighbor *as it appears in the current, not successor, state.*

### 3.3.2 The Bounded Predecessor Existence Problem

Sutner's result establishes the NP-Complete membership of the problem of determining a bounded-support immediate predecessor $X$ for a given $Y = AX$ for a fixed universal rule $A$, immediate predecessor configuration $X$, and fixed successor configuration $Y$ over the field $\mathbb{F}_2$ and undirected, closed, connected graph $G$.

In the BPEP, the instance is an additive cellular automaton $(G, \mathbb{F}_2, \rho^+)$, an immediate successor $Y$, and a bound $\beta$. The yes or no question format for this problem is as follows:"Is there a configuration $X$ such that $\rho(X) = AX = Y$ and $\|X\|_0 \leq \beta$?" This leads to the following theorem:

**Theorem 3.10.** *(Sutner) The Bounded Predecessor Existence Problem (BPEP) is NP-complete for $\mathbb{F}_2^+$-automata on finite graphs. The problem remains NP-complete even if the successor configuration is fixed to be* $1$, *that is,* $1(v) := 1$ *for all* $v$ *in* $V$.

*Proof.* The following proof utilizes the same presentation, technique, and illustration from (9), but is given additional clarification and exposition beyond the source material.

Since matrix multiplication is performed in polynomial time, it follows that given a guessed affirmative instance $X$ to the BPEP, $X$ can be verified to be of appropriate size, and $AX$ can be computed to equal $Y$, in polynomial time. Hence the BPEP is in the class NP.

The three-satisfiability (3SAT) problem, one of Karp's 21 NP-Complete problems (4), is now reduced to BPEP. 3SAT asks whether a boolean formula $\theta = \phi_1 \wedge \ldots \wedge \phi_m$ with clauses $\phi_j = z_{j,1} \vee z_{j,2} \vee z_{j,3}$, where $z_{j,k}$ are literals (that is, they can be either $x_k$ or $\bar{x}_k$) over $n$ varables in $X = \{x_1, \ldots, x_n\}$, is satisfiable.

3SAT can be imagined as a special case of SATISFIABILITY as given in its matrix form in Proposition 3.2. In the case of 3SAT, the representative matrix $A$ has $m$ rows corresponding to clauses $\phi_1 \ldots, \phi_m$, $n$ columns corresponding to each variable in $X$, and exactly three nonzero elements in each row, with $1_{j,i}$ appearing if and only if $x_i$ is represented in $\phi_j$ and similarly for $-1_{j,i}$ and $\bar{x}_i$.

Under the arithmetic given for the matrix representation for SATISFIABILITY, the same assignment for $x \in \{-1, 1\}^n$ (here, $x$ means a vector, not a Boolean variable) is sought such that each row $j$ of $A$ has at least one column index $i$ such that the element $A_{j,i} = x_i$. This occurs successfully if and only if $Ax = \mathbf{1}$, as in Section 1. As before, the alphabet $x_1, \ldots, x_n$, here meaning row elements of the vector $x$, is numerically represented as $1_1, \ldots, 1_n$.

$-1_i$ stands for the negation of the variable $1_i$; all variables together with their negations form $2n$ *literals* over $X$. A truth assignment for $1_i$ corresponds to placing 1 in $x_i$ and an assignment of false for $1_i$ (meaning $-1_i$ is assigned as true) corresponds to placing $-1$ in $x_i$.

First, a graph $G$ with vertices $1_1, \ldots, 1_n, -1_1, \ldots - 1_n$ is constructed with edges $1_i, -1_i \in E$ for all $i = 1, \ldots, n$ and with no other edges between these $2n$ literals.

Additionally, for each row $j$ of $A$, $j = 1, \ldots, m$, there exists a connected component of $G$, $H_j$, which is itself connected to some of the truth-assigning variables in the previous paragraph, but is not connected to any other components.

For each $H_j$, there exists three a-vertices (standing in for the nonzero entries of row $j$ of $A$) and seven b-vertices (which are used creatively to assign truth values to some, but not all, of the elements $a$). These comprise the ten vertices of $H_j$.

For each $H_j$, exactly one edge exists between $a_{j,v}$ and $b_{j,v}$, $v = 1, 2, 3$; additionally, the edges $b_{j,4}a_{j,1}$, $b_{j,4}a_{j,2}$, $b_{j,5}a_{j,1}$, $b_{j,5}a_{j,3}$, $b_{j,6}a_{j,2}$, and $b_{j,6}a_{j,3}$ exist, meaning three $b$ elements

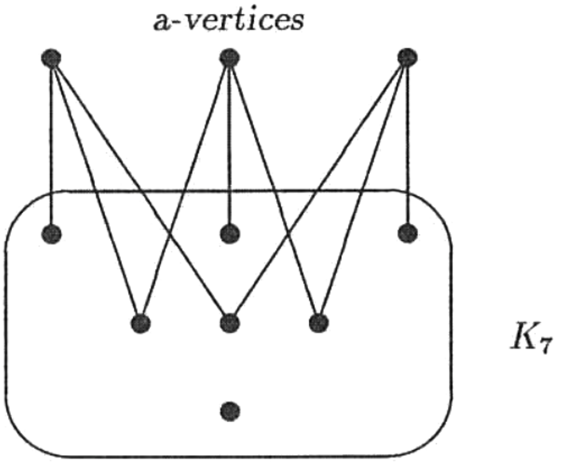

Figure 3.3 A generic $H_j$-component of the constructed automata (9).

are connected to exactly one $a$ element, three other $b$ elements are connected to exactly two of the three $a$ elements, and the seventh remaining $b$ is connected to no $a$ elements. Furthermore, every b-edge is connected to every other b-edge (thus the b-edges form what is known as a $K_7$ graph).

Finally, there is an edge between $a_{j,v}$ and $z_{j,v}$, where each connected $z_{j,v} = 1_{j,i_v}$ or $-1_{j,i_v}$ in accordance to the value appearing in $A_{j,i_v}$, where $\{i_v\} = \{i_1, i_2, i_3\}$ are the column indices corresponding to the three nonzero elements in row $j$ of $A$.

In other words, the three $a_{j,v}$ are directly connected to the three variables represented in row $j$ of $A$.

The illustration of a generic $H_j$-component of $G$ is given in Figure 3.3.

The successor state to the automata on this graph is fixed at $1(v) = 1$ for every vertex in $G$ and the support for the immediate predecessor configuration $P$, $\beta$, is initialized as $\beta = m + n$. The job is to show that if we assume a predecessor state $X$ exists for the all-ones setting, then a truth assignment satisfying $Ax = \mathbf{1}$ exists, and vice versa.

For such a predecessor $X$ to have the immediate successor state 1 at every vertex $w$ of this graph, $|X \cap \Gamma^+(w)|$ (the vertices set to 1 in the predecessor that are adjacent to the vertex $w$, including $w$ itself if its predecessor state is 1) *must equal an odd number* for every vertex in the graph, since the underlying field is $\mathbb{F}_2$ (that is, successor states for any vertex are 1 if and only if an odd number of 1's are added from the vertex's neighbors to determine the successor state).

First, since the vertices corresponding to each $1_i$ and $-1_i$ are connected and no other vertices apart from some $a$ vertices are connected to either of these elements for all $i \in [n]$, it follows that at least $n$ $1_i$, $-1_i$ or $a$ vertices must be set to 1 in the predecessor to obtain a 1 in each vertex $1_i$ and $-1_i$ for all $i \in [n]$.

Additionally, if $b_{j,7} = 0$ in $X$, since the successor state value $\mathbf{1}$ $(b_{j_7}) = 1$, an odd number of $b_{j,t}$, $t = 1, \ldots, 6$, must be set to 1 in $X$. Similarly, if $b_{j,7} = 1$, an even number (or zero) of $b_{j,t}$, $t = 1, \ldots 6$ must be set to 1 to obtain a value of 1 for $b_{j,7}$ in the all-ones immediate successor state. It follows that at least one $b$ in each $H_j$, $j \in [m]$, must be set to 1 in the immediate predecessor state. Since at least $n$ 1's in the predecessor state must be assigned to either $a$ or $1_i$ or $-1_i$, $i \in [n]$, it follows that exactly $m$ 1's are available to assign to the $b$ vertices in each of the $m$ $H_j$ components; therefore exactly one $b$-vertex is assigned 1 and the rest are assigned 0 in each $H_j$.

It follows that exactly $n$ 1's are assigned to either $1_i$, $-1_i$, or $a_{j,v}$ $v = 1, 2, 3$, $j \in [m]$. If a 1 appeared in any $a_{j,v}$, then the corresponding vertex $b_{j,v}$ are connected to an even number of vertices containing a 1 in the predecessor setting, meaning $b_{j,v} = 0$ in the successor state, a contradiction. Therefore no $a$ vertices are set to 1 in the immediate predecessor; hence the remaining $n$ 1's are assigned to either $1_i$ or $-1_i$ (but not neither or both, or else $1_i = -1_i = 0$ in the immediate successor since no $a$-vertices are set to zero).

The placement of these $n$ 1's in the immediate predecessor state among the $1_i$ and $-1_i$, $i \in [n]$ must be such that at least one $a$-vertex connects to a $z_i \in \{1_i, -1_i\}$ set to 1 in the predecessor, for *every* $H_j$ component. This is because exactly one $b$-element of $H_j$ is set to 1 in the predecessor state, and at most two $a$-vertices are connected to any $b$-vertex in every $H_j$. Since all three $a$-vertices in each $H_j$ are set to 1 in the immediate successor state, it follows that at least one $a$-vertex in each of the $m$ $H_j$ components is connected to an element $1_i$ or $-1_i$ set to 1 in the predecessor state. The all-ones successor state therefore attains.

Since the $a$-vertices connect only to values appearing in row $j$ of $A$, then for each clause $j$, select exactly one $a_{j,v}$, $v \in [3]$, that is not connected to the $b$-vetex set to 1 in the predecessor configuration, from which it follows that this $a_{j,v}$ must be set to connected to a $1_i$ or $-1_i$ set to 1 in the predecessor state. Set row $x_{i_v}$ of $x$ equal to 1 if $a_{j,v}$ shares an edge with $1_{i_v}$, or set $x_{i_v}$ of $x$ equal to $-1$ if $a_{j,v}$ shares an edge with $-1_{i_v}$. By definition of the $a$-vertices, it must be the case that $A_{j,i_v}$ matches the value in $x_{i_v}$. Set any remaining elements of $x$ to be 0. Since each row $j$ has exactly one such matching, it follows that $Ax = \mathbf{1}$, showing that $x$ corresponds to a satisfiable setting for the logical formula $\theta$.

Conversely, if $\theta$ is satisfiable, simply select $n$ vertices in the underlying graph of $1_i$, $-1_i$ components as above, placing 1 in the variables representing a satisfying formula for $\theta$.

Place all remaining $m$ 1's in the $b$-components of the $m$ $H_j$ that connect to the one or two $a$-components not connected to a $1_i$ or $-1_i$ set to 1 (at least one must be connected, by assumption of a satisfiable setting). If all three are connected to a variable set to 1 already, place the 1 for $H_j$ in $b_7$. The resulting immediate successor then has all 1's, as required. The NP-Complete problem 3SAT is reducible to instances of BPEP through a construction of a graph whose size is a polynomial function of the size of clauses in $\theta$. Since a proposed solution for 3SAT can be checked in polynomial time, we are done. ////

### 3.3.3 NP-Hardness of Problem 1.2 Over $\mathbb{F}_2$

**Theorem 3.11.** *Problem 1.2 is NP-Hard over the field $\mathbb{F}_2$.*

*Proof.* We embed BPEP as follows. Let $(G, \mathbb{F}_2, \rho)$ be an automata on the graph $G$.For the automata $(G, \mathbb{F}_2, \rho)$, construct the $m \times m$ (symmetric) incidence matrix $A$ as defined in the previous section. Since the rule $\sigma^+$ is assumed, the elements on the diagonal of $A$ are all $1's$. Set $b$ to be the $\mathbb{F}_2^m$ vector of all $1's$. Assume a predecessor state to the state of all 1's exists for this automata; it follows that a minimal predecessor state exists to the state of all 1's. Set $x \in \mathbb{F}^m$ to be the values corresponding with the (enumerated) $m$ vertices of $G$. It follows that $Ax = \mathbf{1}$, and if a solution that is smaller than $\|x\|_0$ exists for this system, setting the vertices of the graph to these values produces the all-ones state as an immediate successor, contradicting the assumption that the minimal setting has been found. Therefore, $x$ is a solution of minimal support for $Ax = 1$.

On the other hand, suppose $x \in \mathbb{F}_2^n$ is a minimal-support solution to the system $Ax = \mathbf{1}$ where $A$ is the same incidence matrix for the same graph as before. Set the state of each vertex $v_k$ of $G$ to be 0 if $x_k = 0$ and 1 if $x_k = 1$; $x$ assumes no other values due to the construction of $A$ and $b$. Then the value of the state of vertices $v_k$ of $G$ in the successor state of the automata is the sum (in $\mathbb{F}_2$) of all states in $\gamma^+(v_k)$; this summation is exactly represented by $A_{\{k\}} * x$. Hence, since $Ax = b$ where $b$ is the vector with all $1's$, all $n$ successor states of the vertices of $G$ is 1. If a smaller immediate predecessor setting than this exists for $G$, the corresponding state vector $x$ is smaller than the vector of minimal support to $Ax = \mathbf{1}$, a contradiction.

Therefore, a "yes" instance occurs for $Ax = \mathbf{1}$ if and only if a "yes" instance occurs for the BPEP (specifically, the smallest possible bounded predecessor) on the corresponding graph and automata over $F_2$. BPEP is therefore embedded as instances of $Ax = b$ over $\mathbb{F}_2$, proving Problem 1.2 is in the class NP-Hard over the field $\mathbb{F}_2$. ////

## 3.4 Conclusion

This chapter explores the computational complexity of solving the central problem of the thesis. In Section 1, we discover that Problem 1.2 is NP-Hard over all fields by extending a result in the literature showing NP-Hardness over the real and complex fields;

new and reformulated results in terms of linear algebra conclude the first section. In the second section, a result showing the NP-Hardness of the problem represented by the $\ell_q$ minimization process for $0 < q < 1$ is obtained, and the same proof is discovered to be surprisingly effective in showing NP-Hardness of Problem 1.2 over the real and complex fields from a new perspective, demonstrating the similarities in the two processes and revealing deep, connected aspects in both problems that directly relate to their respective difficulty.

The third and final section takes another unique angle, connecting the case of Problem 1.2 over $\mathbb{F}_2$ with finite-state automata on graphs, and proving the NP-Hardness of this case by reducing the Bounded Predecessor Existence Problem to an instance of Problem 1.2 over $\mathbb{F}_2$. Ramifications of these results are discussed throughout the chapter, and the implications of this chapter's results carry over to the discussion, development, design, and proof of the algorithms and theory in the final two chapters - particularly with the necessary intractability of any program seeking to do the job Algorithm 4.2.2 assumes. An important question - that of connecting the NP-Hardness from Theorem 3.11 to NP-Hardness over any field - is isolated as an important point of exploration in the fifth chapter.

# CHAPTER 4. ALGORITHMS FOR MINIMAL SUPPORT SOLUTION RECOVERY

This chapter focuses on algorithm-based recovery of minimal support solutions to consistent underdetermined systems.

We first provide a brief survey of the discipline of *Compressive Sensing*, concluding with an analysis of finite field compressive sensing. Basis Pursuit, explored in previous chapters, is given formal footing here, including a discussion of conditions for guaranteed minimal support recovery. The greedy Orthogonal Matching Pursuit (OMP) method is analyzed briefly, and the first section concludes with an adaptation of OMP to systems over finite fields.

In the second section, a thorough presentation and analysis of the main algorithm of the dissertation, Algorithm 4.2.2, is provided. The algorithm is given as pseudocode, and then is described informally; the proof of the algorithm's effectiveness and ability to halt in a finite number of steps concludes the second section and ends the chapter.

## 4.1 Compressive Sensing

*Compressive Sensing* is the practice of recovering a sparse *signal* $x \in \mathbb{F}^n$ from an *observation* $b \in \mathbb{F}^m$ of $m$ linear *measurements*, where $m < n$. Rows of the *measurement matrix* $A \in \mathbb{F}^{m \times n}$ are also referred as measurements; the context should be clear.

Compressive Sensing often focuses on the two strategies of *unique minimal support recovery* and *polynomial time approximation*. Applications including wavelets (10), statistical theory (11), and approximation of minimal support solutions for situations when the measurement observation vector $b$ is tampered with *noise* (12). Numerical analysis provides techniques to improve upon known bounds and error encountered by approximation algorithms.

First, compressive sensing results are examined, followed by a brief overview of technique. Although the discipline appears in literature at least as far back as 1979 (5), most related work, including the coining of the term for the discipline itself, was sparked by Donoho in (11) and has been produced within the previous two decades.

Most of what follows in this chapter's first section relies mainly on the exposition in (12), except where otherwise noted.

**Definition 4.1.** A vector $x \in \mathbb{F}^n$ is *s-sparse* if its weight $\|x\|_0 \leq s$, $s \in \mathbb{N} \cup \{0\}$.

Useful properties emerge when the sparsity of the signal is low relative to other parameters present in the given system. One significant result is given in Theorem 2.13 in (12) below.

**Theorem 4.2.** *The following are equivalent for $A \in \mathbb{F}^{m \times n}, m < n$:*

*(a) If $Az = Ax$ and $z$ and $x$ are both s-sparse, $z = x$.*

*(b) $ker(A)$ contains no 2s-sparse vectors other than the zero vector.*

*(c) Each map corresponding to the submatrix $A_S$ with $S \subset [N]$ and $|S| \leq 2s$ is injective.*

*(d) Each set of $2s$ columns of $A$ is independent.*

*Proof.* We expand on the proof in (12), and present each portion of the proof with full rigor below.

(b) implies (a): let both $x$ and $z$ be s-sparse. Let $Ax = Az$, implying $Ax - Az = A(x - z) = 0$, meaning $x - z$ is in the null space of $A$. Since both $x$ and $z$ are assumed s-sparse, we have that $\|x - z\|_0 \leq 2s$; that is, $x - z$ is 2s-sparse. By the assumption in part (b), $x - z = 0$, meaning $x = z$. Hence $Ax = b$ has a unique $s$-sparse solution.

(a) implies (b): Suppose $v$ is in the null space of $A$ and $\|v\|_0 \leq 2s$. Then we can write $v = x - z$ for some vectors $z$ and $x$ that are $s$-sparse and such that the support of $z$ and $x$ are disjoint. We have, from the assumption about $v$, that $Av = A(x-z) = 0$ and hence by linearity that $Ax = Az = b$ for some $b \in \mathbb{F}^m$. But the assumption given in part (a) implies that $x = z$ since $x$ and $z$ were assumed to be $s$-sparse. Since $x$ and $z$ are assumed to have disjoint support and are equal, it follows that $x = 0$ and $z = 0$, and hence that $v = x - z = 0$.

(b) holds if and only if (c) holds: let $A_S$ be an arbitrary submatrix of $A$ with $|S| = 2s$ columns. $A_S$ is injective if and only if $A_S v_S = 0$ implies $v_S = 0$. For all $v$ of support at most $S$ such that $|S| = 2s$, letting $v_S$ equal the values of $v$ on the indices of support $S$, the latter statement then holds if and only if $Av = A_S v_S = 0$ implies $v = 0$ for all such $v$. This is the case if and only if the null space of $A$ only has 0 as a 2s-sparse vector.

(c) holds if and only if (d) holds: Again, let $v \in \mathbb{F}^n$ be equal to $v_S \in \mathbb{F}^{|S|}$ on the rows indicated by $S$ and 0 otherwise. Each map corresponding to $A_S$ with $S \subset [N]$ and $|S| \leq 2s$ is injective if and only if $A_S v_S = 0$ implies $v_S = \mathbf{0}$. This holds if and only if

$$Av = \sum_{k \in S} v_k \mathbf{A_k} = \mathbf{0}$$

implies $v_k = 0$ for all $k \in S$, establishing independence of columns $\mathbf{A_k}$ of $A$ for all $k \in S$. Since $S$ was selected arbitrarily, every $2s$ set of columns of $A$ are independent. Hence (c) holds if and only if (d) holds. ////

The next theorem deals with sparse signals encoded by underdetermined linear systems over the real and complex fields. The crux of the theorem is that if we *fix* the signal $x$, we discover that there is a matrix $A$ of relatively small row rank such that the signal is *uniquely* recoverable from its corresponding measurement vector $b$. The proof allows for a possible selection of many such matrices; the theorem implies that matrices with random vectors drawn from the unit sphere can uniquely encode a signal $x$ with probability 1 (though approximated polynomial-time recovery methods may not be guaranteed).

**Theorem 4.3.** *(12) For any $N \geq s+1$, given an $s$-sparse vector $x \in \mathbb{C}^n$, there exists a measurement matrix $A \in \mathbb{C}^{m \times n}$ with $m = s+1$ rows such that the signal $x$ can be reconstructed from the measurement vector $b = Ax$.*

The proof of Theorem 4.3, though not included here, relies on matrices not selecting entries from a set of *Lebesgue measure* (7) zero. Though not directly related to the original work in this chapter, any implementation of Algorithm 4.2.2 should take advantage of random measurement matrix generation as highlighted in Theorem 4.3 for purposes of expedient and comprehensive empirical demonstration.

Current approaches to solving Problem 1.2 center on average-case polynomial time approximation algorithms; the two most common methods, Basis Pursuit and Orthogonal Matching Pursuit, are given in this section.

Though a significant focus on controlling an acceptable tolerance of noise in the receipt of the measurements persists in Compressive Sensing techniques, addressing recovery for noisy measurements is beyond the purpose of the dissertation, so these techniques are not discussed. Likewise, techniques that are specific to both statistics and Fourier analysis are not assessed here, though acknowledgement of the presence of a large body of work dedicated to these areas is due; in particular, the discipline of Compressive Sensing originally arose from exploring techniques to store large, but sparse, vectors of Fourier coefficients (11) (10).

We therefore move forward with analysis of the popular Basis Pursuit and Orthogonal Matching Pursuit (OMP) algorithms, which are the two most common Compressive Sensing algorithms as well as the two algorithms most relevant to theory presented throughout this dissertation. Basis Pursuit fulfills the intuition gleaned from the examples and informal discussion of $\ell_1$ minimization in Chapter 2; OMP motivates the strategy involved in $F^2$OMP for finite field recovery, an algorithm whose formulation and analysis completes this overview and whose techniques bear similarity with portions of Algorithm 4.2.2.

A third class of algorithms, known as *threshholding* algorithms, are based on actions of the adjoint $A^*$ to the measurement matrix $A$; these algorithms, though important within Compressive Sensing, share no aims with the goals of this work and, as such, are not covered in this section.

### 4.1.1 Basis Pursuit

The Basis Pursuit algorithm was developed by Donoho (11) and Candes and Tao (10), originally for recovering encoded sparse Fourier coefficients. The algorithm is generalized in (12) as follows:

**Algorithm 4.1.1** Basis Pursuit

```
Input: (Measurement matrix $A$, measurement vector $b$)
Output: A minimal support vector $z \in \arg \min_{x:Ax=b} (\|x\|_0)$
Instructions: Minimize $\|x\|_1$ under the constraint $Ax = b$.
```

Algorithm 4.1.1, though not explicitly stated as an algorithm here or in the source material from (12), can be reformulated as a linear programming problem addressed by the Simplex Method (as shown in Example 2.9), meaning that the problem of finding $\ell_1$ minimizers can be addressed by tractable processes such as the slightly adjusted Simplex Method presented in (22).

### 4.1.2 Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) discovers, at each iteration, the column of $A$ closest to the observation $b$, projects $b$ onto the span of all columns selected, and then repeats the process, updating $b$ to its corresponding projection vector or *residual*, and halting when the currently loaded residual is the zero vector.

OMP is not presented in this fashion in the literature, as the above setup mucks through $\mathbb{R}$ for each remaining column of $A$ in Step 1, along with traversing the whole of $\mathbb{R}^n$ to satisfy Step 3.

**Algorithm 4.1.2** OMP

```
Input: (measurement matrix $A$, measurement vector $b$)
Output: Minimal support vector $x$ to $Ax = b$.
Instructions:
Initialize $S = \varnothing$, $x = \mathbf{0}$, $b^{(1)} = b$, and $t = 0$.
Step 1:
    $t = t + 1$.
    Find the column $A_k$ of $A$ and $\alpha_t \in \mathbb{R}$ that minimizes $\|b^{(t)} - \alpha_t A_k\|_2$ over all columns $k \notin S$.
Step 2:
    Place the index of support of a column meeting criteria in Step 1 into the support vector $S$.
Step 3:
    Calculate the vector $z$ that minimizes $\beta = \|b^{(t)} - A_S z\|_2$.
Step 4:
    If $\beta = 0$, return $x$ such that $x_{S_k} = z_k$ for all $k \in S$ (and by initialization, zero otherwise).
    Else, let $b^{(t+1)} = b^{(t)} - A_S z$ and return to Step 1.
```

In practice, the above representation is overdramatic, since $\|b^{(t)} - \alpha_t A_k\|_2$ is minimized by $\alpha_t = \frac{<b^{(t)},A_k>}{\|b^{(t)}\|_2^2}$ for the column $A_k$, $k \notin S$, of maximal inner product with $b^{(t)}$ for all such columns, and $z$ in Step 3 is the vector inducing the projection vector from $b^{(t)}$ onto the subspace spanned by $A_S$; since $b^{(t)}$ is iterated to be the projection vector after the previous loop, it is perpendicular to the span of $A_{\{S \setminus k\}}$; hence, it follows directly that $z_j = \alpha_j$ for all $j \in S$. The purpose of formatting Algorithm 4.1.2 in this way is to connect this process with the algorithm $F^2$OMP that concludes this first section.

### 4.1.3 Recovery Conditions

A condition difficult to assess is the *null space property;* if each nonzero vector $v$ in the null space of $A$ is such that $\sum_{k=1^s} |v_k|_1 \leq \|v\|_1$ where $v_k$ are the $s$ highest-magnitude entries of $v$, Algorithms 4.1.1 4.1.2 are guaranteed to solve 1.2 and to recover the same solution encoded, so long as the signal is of support less than or equal to $s$.

More handy is the *mutual coherence* property, defined as $\mu(A) = \max_{i,j \in [n], i \neq j} < A_i, A_j >$ . OMP and Basis Pursuit are guaranteed to recover a minimal support solution for $Ax = b$ over the real and complex fields when the mutual coherence of $A$, denoted $\mu(A)$, is such that $\mu(A) < \frac{1}{2s-1}$, with $s$ the sparsity of the encoded signal. $\mu(A)$ is monotone decreasing as a function of $s$, meaning that the maximum $s$ satisfying $\mu(A) < \frac{1}{2s-1}$ is the maximal bound for guaranteed and unique s-sparse support recovery. Such a condition implies that the minimum eigenvalue of $A^*A$, $\lambda_1$, is such that

$$\lambda_1 \geq 1 - (s-1) * \mu(A) > 0,$$

and $\lambda_1 > 0$ implies satisfaction of the Null Space Property condition for exact recovery for the OMP and Basis Pursuit algorithms (12) (20).

### 4.1.4 Finite Field Minimal Support Solution Recovery

We conclude this introductory section with a discussion and presentation of an algorithm for minimal support recovery over finite fields. Though this area of compressive sensing is promising for dictionary learning, algorithmic theorem proving, cryptography, and lossless data encryption, research is sparse when compared to available material addressing systems over the real and complex fields.

This comparative lack of research is possibly due to the inability to employ $\ell_1$ and $\ell_2$ norms on structures over finite fields, thus removing the techniques and the easily assessable bounds (including mutual coherence) that are thoroughly explored in the case of Problem 1.2 over the real and complex fields (13) (14).

Draper et. al. examine methods for the related area of rank minimization of matrices over finite fields in (17). Das and Vishwanath (13) represent $A$ over an extension field of the base field, combining rows of $A$ together as terms of a polynomial of a primitive element in the extension field chosen. Li et. al. employ Bayesian learning for robust recovery over finite fields in (18).

Perhaps the simplest technique related to finite field compressive sensing is employed by Bioglio, Coluccia, and Magli in (14) in a creative adaption of Orthogonal Matching Pursuit to the $\ell_0$ "norm." Bioglio et. al. cite a talk by Draper in 2009 (unavailable to the author) as the source of proof for the effectiveness of techniques employed by this translation of OMP to the finite setting, a technique known as Finite Field Orthogonal Matching Pursuit ($F^2$OMP). The algorithm simply adapts Algorithm 4.1.2 by substituting the $\ell_0$ metric for the $\ell_2$ metric, and proceeding as above with a few minor adjustments.

**Algorithm 4.1.3** F2OMP

```
(14)
Input: (Matrix $A$, vector $b$)
Output: A minimal support vector $y$ for $Ax = b$ or a message of algorithm failure.
Instructions:
Initialize $b^{(0)} = b$, $A^{(0)} = A$, $S = \varnothing$, $y = 0$, $t = 0$.
Step 1:
$t = t + 1$.
Find a column $k$ of $A$ and $\alpha in \mathbb{F}$ that minimizes $\|b^{(t)} - \alpha A_k\|_0$ across all columns with indices not already in $S$.
If $\|b^{(t)}\|_0 \leq 1 + \|b^{(t)} - \alpha A_k\|_0$ for all $\alpha \in \mathbb{F}$ and all remaining columns of $A$ not indexed by $S$, the algorithm fails.
Step 2:
Record the index $k$ into $S$.
Step 3:
Multiply row $i$ of $A$ and $b^{(t)}$ by $(A_{i,k})^{(-1)}$, perform row operations on $A$ and $b^{(t)}$ so that 0 is in all other row elements of column $A_k$, and swap row $i$ with row $t$ for both $A$ and $b^{(t)}$.
Step 4:
If $b_j^{(t)} = 0$ for all $j > t$, set $y_k = b_r^{(t)}$, $r = 1, \ldots, t$, iff $k \in S$ and $A_{r,k} = 1$; return $y$ and end. Else, return to Step 1.
```

Two important aspects of this algorithm are noted. The first is that there is no need in Step 1 to check all elements $\alpha \in \mathbb{F}$; indeed, column $A_k$ equals $b^{(t)}$ at a certain row $i$ if and only if $\alpha A_{i,k} = b_i^{(t)}$ if and only if $\alpha = b_i^{(t)}(A_{i,k})^{-1}$. Hence, only at most $m$ elements $\alpha \in \mathbb{F}$ are needed to check each particular column $A_k$ in Step 1.

Second, as a consequence of the limited selections for $\alpha$ in step 1, the algorithm performs particularly well against other algorithms repurposed for finite field minimal support recovery that require canvassing of all elements in $\mathbb{F}$, particularly when the field is large.

Provided no failure occurs in Step 1, the column that minimizes the number of nonzero elements maximally for all columns of $A$ is selected, with its index placed into $S$ in step 2. A multiple of this column represents the closest vector to $b^{(t)}$ in the sense of minimization of the *Hamming distance* $\|\alpha A_k - b^{(t)}\|_0$, analogous to OMP's minimization of the usual Euclidean distance.

In Step 3, the algorithm multiplies row $i$ of the augmented system by $(A_{i,k})^{(-1)}$ (this gets precisely $\alpha$ in $b_i^{(t)}$) and then Backsolves to get zeros in all other columns in Column $A_k$ via row operations performed on the augmented system $[A|b^{(t)}]$. This gets the number of zeros promised to $b^{(t)}$ in Part 1, except for the zero promised for row $i$.

After this, a row exchange between row $i$ and row $t$ is performed to the augmented system, leaving $e_t$ where Column $A_k$ once was.

In step 4 of $F^2$OMP, if all rows below row $t$ are zero, a minimal support solution is returned by setting $y_k = b_r$ iff $A_{k,r} = 1$ and $k \in S$. The remaining $n - r$ elements of $y$ are zero by the initialization step; therefore, the algorithm returns $y$ and ends. Else, the algorithm returns to Step 1. This reflects Steps 3 and 4 in OMP in that the residual vector in the Hamming sense is represented by the remaining rows of $b^{(t)}$ below row $t$; once these elements are all zero, a solution is achieved by $F^2$OMP, similar to the halting condition for OMP.

Any recovered support vector has at most $d$ nonzero elements, where $d$ represents the loop of the algorithm when the residual vanishes.

In the algorithm we construct in the next section, minimal support solutions are returned iff a similar condition to the residual condition for $F^2$OMP has been identified, and the algorithm guarantees that the minimal support size is *exactly* $d$.

## 4.2 A Novel Algorithm for Complete Minimal Support Recovery over Any Field

In this section, we present a novel algorithm for minimal support solution recovery. The algorithm is first given as pseudocode, is discussed in informal terms, and then is both proven to successfully halt and proven to recover minimal support solutions by induction. To the author's knowledge, this represents the first comprehensive algorithm in the literature for recovering *all* minimal support solutions to underdetermined systems over any field that is not a direct combinatorial assessment of increasingly large numbers of columns.

Following Chapter 3's proof in Theorem 3.6, any comprehensive algorithm, including the algorithm in this section, must necessarily require a number of steps unbounded by any polynomial function of the size of the input in the general case; this, coupled with the fact that compressive sensing often focuses on *unique* signal recovery, is likely why no such comprehensive algorithm has yet been pursued.

Nonetheless, in the spirit of creating relatively efficient exponential solvers of the (NP-Complete) Satisfiability problem (see e.g. (16)), we present Algorithm 4.2.2 both as a first attempt at improving efficiency and time beyond random searching. The algorithm is named MinSup, an abbreviation of "minimal support."

### 4.2.1 Pseudocode of MinSup

Algorithm 4.2.2 represents a function that is part of a larger program, ParentProgram, that forms an augmented matrix representing the system, removes redundant rows, quickly checks for pathological cases, and then calls the main algorithm of this section, Algorithm MinSup. Readoff and Backsolve are subalgorithms utilized by ParentProgram and MinSup, and these two subalgorithms, as well as ParentProgram, are given summary treatment in place of pseudocoding for brevity.

**Algorithm 4.2.1** ParentProgram

Input: $(A \in \mathbb{F}^{m \times n}, b \in \mathbb{F}^m)$
Output: minimal support solutions to $Ax = b$, or an error.

**Step 1:**
Form an *augmented system* $G$, concatenating $b$ after the last column of $A$. The row and column size of $A$ is recorded. $G$ is replaced with its row-reduced form $RREF[G]$.

**Step 2:**
In Step 2, a check is in place to discover an inconsistent matrix.

**Step 3:**
In Step 3, Algorithm 4.2.1 moves zero rows to the bottom of $G$ as these rows account for dependent rows within $G$. These zero rows are then removed from $G$, and $m$ is updated recursively as $m = m - Z$.
The step then identifies if $b = 0$ and halts after returning the unique solution $\mathbf{0} \in \mathbb{F}^{\mathbf{n}}$.

**Step 4:**
In the final step of ParentProgram, the cases $m > n$ and $n = m$ are dealt with in turn. Therefore, any system remaining is necessarily consistent and underdetermined, with $b \neq 0$ and $A$ of full, positive row rank.
CALL $MinSup(G)$.

### 4.2.2 Discussion and Proof of Algorithm 4.2.2

What follows is an informal discussion of Algorithms 4.2.1, 4.2.2, 4.2.3, and 4.2.4. Formal proofs follow this discussion

**Definition 4.4.** The *depth* or *depth level* $d$, $1 \leq d \leq m$, refers to the $d$th iteration of the first While loop in Step 2 of MinSup.

**Definition 4.5.** A *branch matrix* is any matrix initialized or created in any step of Algorithm 4.2.2. By *parent branch matrix* or simply *parent*, we mean a matrix generated during the immediately preceding depth level, and are symbolized $M_v$ or simply $M$. On the other hand, a *child matrix* is a branch matrix created at Step 8 at a certain given depth level; the $M_V$ which led to its generation may be referred as its *parent matrix.*

In Step 1, several variables are initialized before the loops of MinSup begin.

The variable $d$ indicates the depth level. Since any minimal support solution must be such that $\|x\|_0 \leq m$, $d$ is initialized at $m$. If the program first discovers a solution on a row $r < m$ in Step 6 of some depth $r$, $d$ is updated accordingly to halt at the depth $r$.

**Algorithm 4.2.2** MinSup

```
Input: (Augmented System $G$)
Output: All minimal support solutions to the system $Ax = b$
Instructions:
**Step 1:**
$d = m$, $\text{Tset}_0 = \{0\}$, $M_0 = G$, $t = v = r = 0$.
**Step 2:**
WHILE $r < d$
FOR ALL $v \in \text{Tset}_{r-1}$
$B = \varnothing$, $K = \varnothing$, $M = M_v$.
**Step 3**:
IF $M(r, n+1) = 0$
$c = r$
WHILE $p = 0$
$\{c = c + 1$
$p = M(c, n+1)\}$
$q(1 : n+1) = M(r, 1 : n+1)$
$M(r, 1 : n+1) = M(c, 1 : n+1)$
$M(c, 1 : n+1) = q(1 : n+1)$
**Step 4:**
$M(r, 1 : n+1) = (M(r, n+1))^{-1} M(r, 1 : n+1)$
IF $r \neq m$
FOR $i = (r+1)$ to $m$
IF $M(i, n+1) \neq 0$
$M(i, 1 : n+1) = M(i, 1 : n+1) - M(i, n+1) M(r, 1 : n+1)$
**Step 5:**
FOR $k = 1$ to $n$
IF $M(r, k) \neq 0$
$B = B \cup \{k\}$
**Step 6:**
FOR ALL $k \in B$
IF $M(r+1 : m, k) = 0$ OR $r = m$
$d = r$
$K = K \cup \{k\}$
**Step 7:**
IF $K \neq \varnothing$
FOR $k \in K$
N=Backsolve($M, r, k$)
Readoff($N$)
**Step 8:**
IF $r \neq d$
FOR ALL $k \in B$
$t = t + 1$
$\text{Tset}_r = \text{Tset}_r \cup \{t\}$.
$M_t :=$ Backsolve $(M, (r, k))$
end of parent While loop
```

**Algorithm 4.2.3** Readoff

Input: Augmented matrix $M$ such that identity columns $e_1, \ldots, e_r$ are present in $M$ and, if called from MinSup, $M$ has 0 in rows $r+1$ to $m$ of column $n+1$.

Output: The minimal support solution $x$ is displayed directly and is formed based on the index of identity columns present in $M$.

**Step 1:**

Ready an array preset to zero in each entry to collect column indices corresponding to values in the $n+1$ column to be placed into $x$ at these respective indices.

**Step 2:**

Find the index of each identity column in the matrix, and place the index of all such columns $j \in J$.

**Step 3:**

Change each row $j \in J$ of $x$ to the entry $i$ of the $n+1$ column of $M$ where $i$ is the row on which 1 appears in column $j$ of the matrix.

DISPLAY (x)

**Algorithm 4.2.4** Backsolve

Input: Augmented system $M$, pivot position row $i$ and column $j$.

Output: Augmented system $M$ multiplied by row operations such that column $j$ is changed into the identity column $e_i$ with 1 in row $i$ and 0 in all other rows. This matrix is returned to the program which called Backsolve.

**Step 1:**

The first step multiplies row $i$ by the inverse of the pivot element $M_{i,j}$ to make sure a 1 is present in position $i, j$ of $M$.

**Step 2:**

This puts 0 in every entry in column $j$ except for the 1 in position $i, j$. This means row operations have transformed the sent matrix $M$ into a matrix whose $j$th column is column $e_i$ of the identity matrix $I_{m \times m}$.

RETURN(M)

In general, Tset$(r)$ is the set of indices $t$ of the child matrices $M_t$ constructed exactly at depth $r$. These variables inform MinSup exactly which matrices $M_v$ to load and examine at the next depth level, $r + 1$. $M_0$ is initialized as $G$. Since each $M_t$ inherit the row exchanges possessed by its parent $M_v$ with no more than one additional row exchange, the order in which $t \in \mathit{Tset}(r-1)$ is accessed at the subsequent depth level does not matter.

Upon execution of the parent While loop beginning Step 2, $r$ is immediately iterated $r = r + 1$. The variables $v \in \text{Tset}(r-1)$ is used to load $M_v$ into the local variable $M$; the program searches for a solution, or else children for the next loop.

Tset$(r)$ is then formed and initialized as empty for the newly initialized current depth, and then the set Tset$(r-1)$ containing the indices $v$ of all parent matrices $M_v$ constructed in the previous overall loop is loaded and commences the main heavy work of the new depth level. Step 2 completes by initializing the variables $B$ and $K$ to be empty.

Step 3 checks whether $M_{r,n+1} = 0$ and, if so, finds the nearest row underneath row $r$ where an element of column $n+1$ is nonzero, and then swaps these rows. Such a nonzero is guaranteed to exist, or else the matrix $M_v$ currently loaded from the immediately previous depth level would have triggered a solution in a previous depth.

Step 4 begins the crucial phase of MinSup. First, row $r$ is multiplied by $(M_{r,n+1})^{-1}$, (after row exchanges performed by Step 3 to clear any possible zero in that position). This obtains a 1 in position $r$ of column $n + 1$.

Step 5 then uses row operations on the system, pivoting on row $r$ of column $n + 1$ to obtain 0 in rows $r + 1$ to $m$ of column $n + 1$ of the matrix currently loaded in $M$.

Rows above $r$ throughout the tableau are left unaffected by Steps 4-5.

After row operations are performed to turn column $n+1$ into $e_r$ for rows $r$ to $m$, the program in Step 5 checks every one of the first $n$ columns of the (now modified matrix) $M$ and records into the set $B$ the index of every column with a row $r$-nonzero entry. For all columns whose indices are recorded in $B$ in Step 5, Step 6 first checks all rows beneath $r$ and identifies all columns that fail to have at least one nonzero below; if so, the program records these indices into the set $K$. If $K$ is discovered to be nonempty, the minimal support solution depth $d$ is downgraded from its initialization as $m$ to the current loop $r$ if needed (note, the phrasing earlier in this paragraph imply vacuously that Steps 6 and 7 are triggered automatically if depth $m$ is reached).

Each index $k \in K$ (if any) is accessed and the corresponding solution is reported by Algorithm 4.2.3. The program continues forward with the rest of the matrices indexed by Tset$(r-1)$, repeating the Step 6 and Step 7 process for all remaining qualified matrices at this depth $r$ if necessary, generating additional $d$-sparse solutions if any. The program therefore halts after matching $r$ with $d$ at the restart of the main WHILE loop.

On the other hand, if $K = \varnothing$, Step 7 is skipped, and the set $B$ built in Step 5 is accessed by Step 8 if no minimal support solution has yet been found from a previously analyzed parent matrix at the current depth.

For all columns with index $k \in B$, the program forms a new child matrix $M_t$ by calling Backsolve onto $M$ and its intended pivot column $k$, which it row-reduces to obtain $e_r$.

After these new children are made for the particular parent matrix loaded, or if a previously loaded matrix has triggered a response for a solution, if Tset$(d-1)$ still has indices for unexamined matrices, the program returns to the relevant part of Step 2 and continues this process until Tset$(d-1)$ is exhausted.

If no minimal support solution is found at the current depth level, the program loops again, accessing the next depth level $r+1$ at Step 2, and continues until a minimal support solution is reached at or prior to the initialized maximum possible depth level $m$. This ends the informal discussion of Algorithm 4.2.2 and its accompanying programs.

### 4.2.3 Proof of MinSup

In this section, the algorithm informally given above is proven. We begin with some lemmas and observations needed by the more difficult proofs that conclude this section.

**Observation 4.6.** $Ax = b$ is inconsistent if and only if a row of $RREF(G)$ is zero in column positions 1 to $n$ and nonzero in column position $n + 1$.

**Lemma 4.7.** The parent program halts without calling MinSup if and only if an inconsistent system, an overdetermined system, or a system with a unique solution is placed into the program.

**Observation 4.8.** ParentProgram returns the minimal support solution to all systems with a unique solution.

**Observation 4.9.** $Ax = b \neq 0$ has at least one solution, $b \neq 0$, and $A \in \mathbb{F}^{m \times n}, 0 < m < n$ is row-independent with rank greater than zero for all systems accessed by MinSup within the parent program.

**Observation 4.10.** Algorithm 4.2.2 presents output only in Step 7.

We call a matrix $M_v$ that triggers Steps 6 and 7 (and, as this section shows, a minimal support solution) a *completed branch*.

**Lemma 4.11.** Algorithm MinSup has a finite number of operations for each step at any given depth level.

It is easy to see that a vector solving each child matrix as adjusted and analyzed in Steps 5-8 at depths $d$ and below also solves $Ax = b$, as all these steps and all these depths involve only row modification of the augmented matrix $M_v$'s ancestors at every step of every depth, including $M_0$, the original system.

**Proposition 4.12.** A solution solving a given completed branch $M_v$ as reported at Step 7 at an arbitrary depth level also solves $Ax = b$. Moreover, this same solution solves each of $M_v$'s predecessor parent matrices at every depth and step.

More difficult to show is that each solution of weight bounded by $m$ is traced by some sequence of matrices during subsequent iterations of MinSup. We prove this result by induction. The first step is relatively easy, and reveals a benefit to MinSup: exchanging the measurement vector with a basis vector after Step 5 reveals a sort of residual similar to $F^2$OMP from Section 1 in rows $r+1$ to $m$. The hope is to continue iterating the row (and therefore the support) until some branch encounters a lowest depth at which one or more columns lose this residual.

**Lemma 4.13.** All solutions $z$ to $Ax = b$ have minimal support weight 1 iff a row $k$ of $z$ indexes a column $A_k$ of $A$ such that $Az = cA_k = b$ for $c \neq 0$; this occurs iff $M_0$ has no nonzeros beneath row 2 of column $k$ at Step 5 of Depth 1 of MinSup, causing MinSup to halt at Depth 1 with all weight 1 solutions.

*Proof.* Suppose a solution $z$ to $Ax = b$ has support $S$ of weight $\|S\| = \|z\|_0 = 1$ only on the index $k$. By hypothesis $Az = A_S z_S = A_k x_k = b$. If MinSup is analyzing the system, $b$, and therefore $A_k$, cannot be the zero vector; therefore $(A_{1,k})^{-1}b_k \neq 0$ with $z_i = 0$, $i \neq S$ is a minimal support solution (of weight one) returned after Step 7 of depth one of Program MinSup. Since all such weight-one solutions appear in this way, Program MinSup therefore identifies and returns all weight one solutions. ////

Likewise, some other depth $1 < d \leq m$ is the halting depth if and only if minimal support solutions to $Ax = b$ are of weight $d$ and are all discovered during depth $d$. We show this in what follows, continuing the strong induction whose base step is presented by Lemma 4.13 above.

**Lemma 4.14.** Algorithm 4.2.2 halts at some depth $1 < r \leq m$ iff minimal support solutions of weight exactly $r$ exist, are recovered, and are all reported by MinSup.

*Proof.* MinSup halts at $r$ iff Step 6 resets $d = r$ or if $d = m$; in either case, for some child matrix $Mv$, $v \in \text{Tset}(r-1)$, there exists at Step 5 of depth $r$ some column $k$ with a positive row element in $r$ but with no nonzero elements beneath row $r$, the same holding for column $n+1$ at this same stage by the construction formed in Step 5. This means $(M_v)_k$ is a constant multiple of $(M_v)_{n+1}$ from rows $r$ through $m$.

By design, this triggers Steps 6 and 7, which turn column $k$ of $M$ into $e_r$ and report the solution

$$x = \sum_{j=1}^{r} M_{j,n+1} e_{K_j}.$$

Therefore, we have that $\|x\|_0 \leq r$. If $\|x\|_0 < r$, then by strong induction, MinSup should have recovered $x$ at depth $r'$, $r > r' = r - \|x\|_0 > 0$, and halted before commencing depth $r$. Hence solutions $x$ recovered at depth $r$ must be of weight exactly

$$\|x\|_0 = r,$$

which was to be shown.

The same inductive reasoning shows that any solution recovered by MinSup at depth $r$ must be a solution of minimal support (of weight $r$); otherwise, any smaller weight minimal support solution would have been discovered earlier, and MinSup would have terminated prior to reaching depth $r$.

For the existence part of the proof, suppose there exists some solution $y$ to $Ax = b$ such that $\|y\|_0 = r$. Then Lemma 4.13 shows that at least one index of support for $y$ is discovered at Step 5 of depth 1 of MinSup. If $\|y\|_0 = 1$, Lemma 4.13 necessitates that MinSup reports this solution at depth 1, and concludes after completing all tasks at depth 1.

Let $r'$ denote a depth level of MinSup such that $1 < r' < r$, and suppose at the end of Step 5 of depth $r'$ there are no matrices indexed by $\mathrm{Tset}(r-1)$ supported by a proper subset of the support for $y$. By induction, $r'-1$ indices of support for $y$ have been identified, and some matrix $M_{v'}$ with $v' \in \mathrm{Tset}(r'-1)$ exists with corresponding columns row-exchanged for the pivot columns $e_1, \ldots, e_{r'-1}$. At step 5 of depth $r'$, a nonzero entry in column $n+1$ row $r$ of $M_{v'}$ is present, but by the assumption that no remaining indices of support for $y$ are discovered at this level, we have that $(M_{v'}y)_{r'} = 0 \neq c$ where $M_{r',n+1} = c \neq 0$ (after, if needed, any row exchange in Step 4). This contradicts Proposition 4.12.

Finally, assume $r$ is the size of support for $y$. This occurs if and only if some $M_v$, $v \in \text{Tset}(r-1)$, discovers a last remaining index of support for $y$ and reports the corresponding solution at depth $r$ in Steps 6 and 7 of MinSup, by similar reasoning to 4.13. All tasks at depth level $r$ are then completed, and the flag in Step 6 indicates that MinSup halts upon completion of these tasks; Lemmas and Observations 4.9, 4.10, 4.11, and 2.6 fulfill the remaining detail. ////

**Theorem 4.15.** *Over any field, Algorithm ParentProgram (and necessarily, therefore, Algorithm 4.2.2) recovers all minimal support solutions to all consistent nonoverdetermined systems of linear equations, and halts in a finite number of steps for any (dimensionally correct) linear system.*

*Proof.* We first demonstrate that the program concludes in a finite number of steps for any input. There are several cases that cover all possible entries that can be given to the parent program for MinSup.

Case 1: $Ax = b$ is an inconsistent system.

Case 2: $Ax = b$ is an overdetermined system.

Case 3: $A$ is such that $A$ is full rank with $m = n$.

Case 4: $b = 0$.

Case 5: $Ax = b$ is consistent and of full rank, $m < n$, and $b \neq 0$.

The four cases are all addressed in Lemma 4.7. Earlier results show that the set of minimal support solutions in this case exist, are each of support size $1 \leq \|x\|_0 \leq m$, and must be finite in number. Therefore, Algorithm 4.2.2 uncovers at least one minimal support solution of size $d \leq m$, and then successfully halts after all remaining steps at depth $d$ of MinSup are complete by virtue of the halting setting in Step 6, as outlined in Lemmas 4.13 and 4.14. Furthermore, the algorithms themselves do not involve an infinite number of steps, and MinSup executes no more than $o(m^m)$ loops.This covers all possible linear systems faced by Algorithm 4.2.1 and 4.2.2, and so proves that the algorithm within its parent program successfully halts in a finite number of steps for any system. For the comprehensive solution portion, ParentProgram returns solutions to systems involving an invertible matrix or a zero measurement vector $b$ in accordance with Observation 4.8. We therefore assume $A$ is of full, positive row rank, $m < n$, that $b$ is nonzero, and that $Ax = b$ is consistent, prompting MinSup.

Lemmas 4.13 and 4.14 demonstrate that every minimal support solution is bounded by $m$ (hence, the set of such solutions is necessarily nonempty) and are all recovered by MinSup. ////

Theorem 4.15 can also be proven directly by induction on the size of the input matrix. We omit the cases of systems addressed by Algorithm 4.2.1, covered in Observation 4.8, and assume that MinSup has been called on a consistent, underdetermined system $Ax = b$ with $b \neq 0$ and $A$ of full and positive row rank. We also do not address the certainty of full recovery of all solutions, since the argument verifying this fact in the context of the proof that follows is similar to the relevant proofs for this statement that appear above. We also do not reproduce a proof that nonzero elements appear in indices indicated to be of support for a similar reason, and therefore move forward with this second look at the proof for Theorem 4.15 assuming these facts are established.

This second proof is presented because of the particular interest in the $\{M_t\}$ created for Case 2 of the inductive step; in practice, these matrices need not be explored in any particular order, allowing for powerful message passing interface (MPI) code to run each of the children of $M_0$ on systems with separate processors and memory. The proof that follows outlines this elegant process plainly, and provides a formal footing from which the propriety of usage of these high-powered numerical analytic computation techniques. This fact has profound practical implications for future explorations, and as such, further discussion of these techniques is relegated to the next chapter.

*Proof.* The direct (weak) inductive proof of Theorem 4.15 is as follows. In the base case, $A$ is simply one row with $k \in \mathbb{N}$ columns. Since the system is assumed to be consistent with $b \neq \mathbf{0}$, this is a special case of the base case in Lemma 4.13 and the conclusion follows after an identical proof (note $d = m = 1$ in the initialization phase for any such sized matrix).

In the inductive case, suppose MinSup successfully returns a minimal support solution to every consistent underdetermined system $Ax = b$, $A \in \mathbb{F}^{(m-1)\times(n-1)}$. Now suppose the system $Ax = b$ is consistent, underdetermined, and such that $A \in \mathbb{F}^{m\times n}$. As $b \neq 0$, there are two cases:

Case 1: After row operations reducing column $n+1$ of $M_0$ to $e_1$ in the first iteration of the While loop in Step 2, using row swaps if necessary, there is at least one column $A_k$ of $M = M_0$ whose rows $2, \ldots, m$ are all 0. In all such cases, similar to the base case, it is immediate that every minimal support solution is of weight 1 with $x_k = (A_{1,k})^{-1}b_1 \neq 0$ and $x_i = 0$ otherwise for all such $k \leq n$.

Case 2: If no columns in Step 1 with a nonzero first row entry are direct multiples of $b$, the program iterates via the main While loop, sending the matrices $\{M_t\}$ generated in Step 8 at depth 1 to depth $r = 2$. Each column $k$ of $M_0$ with a nonzero first row entry induces a separate $M_t$, $t \in \text{Tset}(1)$, in Step 8 of depth 1.

Each subloop at depth 2 for $M_v$, $v \in \text{Tset}(1)$, is identical in operation to a program call of MinSup of a system with measurement matrix of size $(m-1) \times (n-1)$ and measurement vector of size $(m-1) \times 1$, as depth 2 of MinSup only operates on rows $2, \ldots, m$ of each $M_t$ and affects only at most $n-1$ columns of $M_t$; the iteration at depth 1 that creates $M_t$ zeroes out rows 2 through $m$ of column $k$ of $M_t$ via calling Backsolve in Step 8, and these same row operations create a nonzero multiple of rows 2 through $m$ of column $k$ in rows 2 through $m$ of column $n+1$ of $M_t$.

By the inductive hypothesis, then, Algorithm 4.2.2 successfully computes the minimal support solution to each of these $(m-1) \times (n-1)$ subsystems $M_t$; since the first loop applies row operations to transform some column of each $M_t$ to $e_1$, these further operations do not affect the presence of this index of support. Therefore, the smallest support solutions among all the minimal support solutions for this set of children generated by Step 8 at depth 1 therefore represent the minimal support solutions to the original system, and the theorem is proven.

////

In practice, the depth of the program operates no further than the depth at which the minimal support solution is discovered; all remaining branches that find no solution at this depth of first encounter are discarded, due to the algorithm's order (in Step 6) to halt after completion of all tasks at this depth.

## 4.3 Conclusion

This section provides a brief survey of compressive sensing techniques, and then develops a novel algorithm recovering all minimal support solutions to any consistent underdetermined system. The algorithm is presented in pseudocode, informally analyzed, and then given firm foundation with both a proof of its halting and a proof by induction that it succeeds in performing its assigned duty. Further opportunities for exploration, implementation, and use of Algorithm 4.2.2 follow in the next and final chapter.

# CHAPTER 5. SUMMARY OF RESULTS AND FUTURE EXPLORATIONS

In this chapter, we first summarize the results of this dissertation and then provide opportunities for future research based on the findings in the previous chapters.

## 5.1 Overview of Results

This work proves the NP-Hardness of finding minimal support solutions to underdetermined linear systems over any field, giving the theorem its full formal statement and proof complete with rigorous foundation for the first time in the literature in Chapter 3.

Following analysis of the success and failure of specific examples in Chapter 2 and briefly overviewing basic results and techniques in the new discipline of Compressive Sensing, Chapter 4 presents a novel algorithm designed to recover all minimal support solutions to any underdetermined system over any field, providing a fresh coding template for full recovery upon which further research can improve. We conclude the work in Chapter 4 with a thorough analysis of the new algorithm, complete with a proof that the algorithm halts and a proof that the algorithm succeeds at doing the task it is assigned to do.

More minor results include some important illustrative counterexamples in Chapter 2, a separate and unique proof of the NP-Hardness of Problem 1.2 over the binary field at the end of Chapter 3, a fresh linear-algebraic representation of SATISFIABILITY with a subsequent re-illustration of the NP-Completeness of the 3D-MATCHING problem in this new context, implementation of the pseudocode in Chapter 4, and some brief proofs of other problems that appear in the literature but lack a full, rigorous foundation.

Though Chapter 3 showed the necessary difficulty of an algorithm whose aim is that of Algorithm 4.2.2, the comprehensiveness of the algorithm and its general ability to work over any field may provide a first step to important problems in compression, cryptography, and theorem proving. Since Compressive Sensing focuses on approximation methods for cases that involve a system with a unique minimal support solution, and involves techniques that work only under certain conditions which, as explained in Chapter 4, can themselves be difficult to assess, Algorithm 4.2.2 represents a first attempt at complete recovery independent of any condition (including the base field $\mathbb{F}$), despite the combinatorial challenge that the NP-Hardness of the problem guarantees to such an undertaking.

## 5.2 Further Explorations

### 5.2.1 Combinatorial Analysis of Algorithm 4.2.2

The first task in any further exploration beyond the scope of this dissertation is a full combinatorial assessment of Algorithm 4.2.2.

We define a *brute force algorithm* as an algorithm that discovers minimal support solutions by row-reduction of a random, combinatorial selection of columns of $[A|b]$. Starting by selecting one column of $A$ and moving on to choosing more upon failure of discovery, such an algorithm would succeed in finding a minimal support solution of size $r$ upon successful random selection of the $r$ columns of the system corresponding to the $r$ indices of support for a minimal support solution, row-reducing each of these rows successfully to the identity columns $e_1, \ldots, e_r$, and spotting a corresponding answer in column $n+1$ in a way similar to Algorithm 4.2.3. In short, a brute force algorithm to solve Problem 1.2 is identical to simply continuously guessing the support until the correct support indices are attained, with the marginal benefit of knowing which elements to place in these positions via the row reduction operations. An effective combinatorial analysis of Algorithm 4.2.2 therefore involves an analysis of its performance versus its most obvious opponent in this brute force algorithm, or any algorithm incorporating its tactics within minor improvements to strategy (for example, avoiding permutations of index sets previously checked).

At least one result down this line of analysis is definitive: MinSup identifies solutions of weight one more quickly than a brute force algorithm with relatively high probability.

In what follows, we assume a Step 4 row exchange is not necessary, which involves at most the cost of one row exchange for the case proven below.

**Proposition 5.1.** Algorithm 4.2.2 probably requires fewer row operations to recover a unique minimal support solution of size 1 than an algorithm designed to recover minimal support solutions via brute force.

Since the general case of a linear system likely has $k > 1$ columns with a nonzero first entry, Algorithm 4.2.2 is a better choice than a brute force algorithm in this case.

As Algorithm 4.2.2 loads matrices that are already reduced at each depth level $r > 1$ in its search for a minimal support solution, the algorithm is not required to perform these row operations again, while a brute force algorithm would need to constantly row reduce all of its chosen columns again at a depth level $r > 1$.

Additionally, the discussion regarding parallelizing the program with message passing interfaces (MPI) prior to the direct inductive version of proof for Theorem 4.15 at the end of Chapter 4 provides a high-performance opportunity for Algorithm 4.2.2. Since child matrices produced at a given depth level can be analyzed in any order, multiple processors can be called by MPI when a single system's processing and memory are not sufficient to handle the number of child matrices generated by an early depth of the program, or can simply be called out of interest of speed even if subsequent depths could have been handled serially by a single machine.

MPI implementations are designed to run identical code in parallel over multiple processors with separate memory, and flags can be implemented to communicate with all cooperating processors as to the status of whether any particular processor has found a solution; when such an event occurs, cooperating processors receive commands from a parent routine based on receipt of this flag operation, are sent the depth level at which a solution is found, and then either terminate if the processor is operating beyond the given depth level or terminate at the end of the given depth level if the processor has not yet reached this point. The parent program then collects all minimal support solutions across all cooperating systems and returns them to the user.

As Algorithm 4.2.2 requires a significant amount of storage space, the above implementation may additionally improve on system space limitations, and therefore expand the depth and size of a system upon which Algorithm 4.2.2 can operate.

Though not rigorous, these observations are enough to formulate the following conjecture; its proof would be an ideal first step in expanding original work beyond what has been accomplished in Chapter 4, and for justifying the use of the algorithm above a brute force algorithm when need for guaranteed full minimal support recovery over any field without constraint is required.

**Conjecture 5.2.** With high probability, Algorithm 4.2.2 requires less run time to find a minimal support solution than a brute force algorithm.

Here, "high probability" means that the probability that Algorithm 4.2.2 succeeds in uncovering a minimal support solution in less run-time than a brute force solution approaches 1 as certain conditions approach associated limits. Identifying these conditions and the applicable limits would be part of this exploration.

### 5.2.2 Polynomial-Time Implementation of MinSup

In Chapter 4, the algorithm $F^2OMP$ is presented prior to the discussion of Algorithm 4.2.2. This algorithm runs in polynomial time, but is not guaranteed to return a solution.

If needed, Algorithm 4.2.2 can likewise be repurposed for this situation. Similar to $F^2$OMP, one branch can be constructed, where at each step the index $k$ of the column of the adjusted system that has a nonzero element in row $r$ and the *most* zeros beneath row $r$ among all other columns with a nonzero $r$-row entry can be added to the branch at Step 8. The intuition behind this idea is that a minimal support solution, especially those of particularly small support compared to $m$ or for matrices whose columns are themselves sparse, involve an index of a column of $A$ that is equal to a multiple of $b$ for a large number of rows, with a handful of other columns adjusting a relatively small number of rows of this multiple. If this is the case, then, similarly to $F^2$OMP, this solution of minimal support can be converged upon quickly.

Finally, approximation algorithms like the ones described in the first sections of Chapter 4 can be implemented within the parent program and called if qualifying conditions attendant to these methods are met by the input, leaving Algorithm 4.2.2 as a sort of fail-safe recovery option if a system is received that fails the tests for approximate recovery. This could give an important role to Algorithm 4.2.2 in contexts where retrieval of a minimal support solution is necessary even if desired approximate solution methods fail.

### 5.2.3 NP-Hardness Questions

During the composition of Chapter 3, the proof for Theorem 3.11 was reached before discovery of Theorem 3.6, though the former is placed after the latter in the writing. The challenge of demonstrating NP-Hardness over any finite field beyond $\mathbb{F}_2$ therefore remains without Theorem 3.6, since current literature either cites the NP-Hardness over $\mathbb{R}$ and $\mathbb{C}$ based on the result from (19), or else cites the (unpublished) result from (5) for the case over $\mathbb{F}_2$ and imply that NP-Hardness over any finite field should be in place given the NP-Hardness of the problem over the simplest applicable finite field (see e.g. Sutner (9) and Bioglio et. al. (14)).

Prior to the discovery of Theorem 3.6, we attempted to extend the proof for NP-Hardness of Problem 1.2 to any finite field from the starting point of Theorem 3.11, but found the problem to be more difficult than initially anticipated. Although technically unnecessary after the sweeping implications of Theorem 3.6, first starting with Theorem 3.11 and reaching NP-Hardness over any finite field (or even $\mathbb{R}$ and $\mathbb{C}$) could prove to be a difficult exploration that may yield rich insight into the nature of the problem in general.

Underdetermined linear systems represent many practical and theoretical scenarios encountered in combinatorics and graph theory, among other diverse applications, meaning that a successful combinatorial exposition down the lines presented in this subsection could yield far-reaching and insightful conclusions for many areas of study.

## 5.3 Conclusion

This concludes the summary of results in this dissertation and the suggestions for future research to extend the work presented in this paper. The Pseudocode in Chapter 4 is presented in open hope that modifications (especially for parallelization or approximation) can sharpen it to a cutting-edge tool either useful for fast, probabilistic recovery of data compressed over any field, or else useful for comprehensive minimal support solution recovery, which could aid advancement in lossless data compression and efficient theorem solving in nonassociative algebra and other fields of study.

# BIBLIOGRAPHY

[1] David S. Dummit and Richard M. Foote. *Abstract Algebra*, Third Edition. John Wiley and Sons, Inc., U.S.A., 2004.

[2] Jonathan D. H. Smith and Anna B. Romanowska. *Post-Modern Algebra*. John Wiley and Sons, Inc., U.S.A., 1999.

[3] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*, Second Edition. Cambridge University Press, New York, NY, 2013.

[4] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall Inc., New Jersey, 1982.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability*. Bell Telephone Laboratories, U.S.A., 1979.

[6] Glenn H. Hurlbert. *Linear Optimization: The Simplex Workbook*, Sixth Edition. Springer-Verlag, New York, NY, 2010.

[7] Walter Rudin. *Principles of Mathematical Analysis*, Third Edition. McGraw-Hill Book Co. Singapore, 1976.

[8] Walter Rudin. *Real and Complex Analysis*, Second Edition. McGraw-Hill Book Co. Singapore, 1976.

[9] Klaus Sutner. "Additive Automata on Graphs." Complex Systems, 2: 649-661, 1988.

[10] Emmanuel Candes, Justin Romberg, and Terence Tao. "Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information." *IEEE Transactions on Information Theory* **52** (2006) 489–509.

[11] David Donoho. "For Most Large Underdetermined Systems of Equations, the Minimal $\ell_1$-Norm Solution is also the Sparsest Solution." *Communications on Pure and Applied Mathematics* **59** (2006) 797–829.

[12] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressed Sensing*, Springer Science and Business Media, New York, NY, 2013.

[13] Abhik Kumar Das and Sriram Vishwanath. "On Finite Alphabet Compressive Sensing." *arXiv:1303.3943v1*, 2013.

[14] Valerio Bioglio, Giulio Coluccia, Enrico Magli. "Sparse image recovery using compressed sensing over finite alphabets." *IEEE International Conference on Image Processing*, 2014.

[15] Richard M. Karp. "Reducibility Among Combinatorial Problems." Manuscript, University of California at Berkley, 1972.

[16] Matthew Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zahang, and Sharad Malik. "Chaff:Engineering an Efficent SAT Solver." *DAC '01: Proceedings of the 38th conference on Design automation*, Las Vegas, Nevada, 2001.

[17] Vincent Y.F. Tan, Laura Balzano, and Stark C. Draper. "Rank Minimzation over Finite Fields." *arXiv:1104.4302v4*, 2011.

[18] Wenjie Li, Francesca Bassi, and Michel Kieffer. "Robust Bayesian compressed sensing over finite fields: asymptotic performance analysis." *arXiv:1041.4313v1*, 2014.

[19] B.K. Natarajan. "Sparse Approximate Solutions to Linear Systems." *SIAM J. Comput.* **24** (1995) 227 – 234.

[20] T. Tony Cai and Lie Wang. "Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise." *IEEE Transactions on Information Theory* **57** (2011) 4680 – 4688.

[21] Joel A. Tropp. "Greed is Good: Algorithmic Results for Sparse Approximation." *IEEE Transactions on Information Theory* **50** (2004) 2231 – 2242.

[22] Jonathan Kelner and Daniel Spielman. "A Randomized Polynomial-Time Simplex Algorithm for Linear Programming." *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing* (2006) 51–60.