Matrix sparsification and the sparse null space problem

Lee-Ad Gottlieb * Tyler Neylon †

Abstract

We revisit the matrix problems sparse null space and matrix sparsification, and show that they are equivalent. We then proceed to seek algorithms for these problems: We prove the hardness of approximation of these problems, and also give a powerful tool to extend algorithms and heuristics for sparse approximation theory to these problems.

*Courant Institute, New York University, New York NY, 10012. E-mail: adi@cs.nyu.edu. Author’s work supported in part by NSF grant IIS 0414763.
†Google, 1600 Amphitheatre Parkway, Mountain View CA 94043. E-mail: tylern@google.com.
1 Introduction

In this paper, we revisit the matrix problems sparse null space and matrix sparsification.

The sparse null space problem was first considered by Pothen in 1984 [25]. The problem asks, given a matrix \( A \), to find a matrix \( N \) that is a full null matrix for \( A \) – that is, \( N \) is full rank and the columns of \( N \) span the null space of \( A \). Further, \( N \) should be sparse, in that it contain as few nonzero values as possible. Motivation for the sparse null space problem stems from the fact that it may be used to solve Linear Equality Problems (LEPs) [8]. LEPs arise in the solution of constrained optimization problems via generalized gradient descent, segmented Lagrangian, and projected Lagrangian methods. Berry et al. [4] consider the sparse null space problem in the context of the dual variable method for the Navier-Stokes equations, or more generally in the context of null space methods for quadratic programming. Gilbert and Heath [15] noted that among the numerous applications of the sparse null space problem arising in solutions of underdetermined system of linear equations, is the efficient solution to the force method (or flexibility method) for structural analysis, which uses the null space to create multiple linear systems. Finding a sparse null space will decrease the run time and memory required for solving these systems. More recently, it was shown [33, 24] that the sparse null space problem can be used to find correlations between small numbers of times series, such as financial stocks. The sparse null space problem is known to be NP-Complete [8], and only heuristic solutions have been suggested for it [8, 15, 4].

The matrix sparsification problem is of the same flavor as sparse null space. One is given a full rank matrix \( A \), and the task is to find another matrix \( B \) that is equivalent to \( A \) under elementary column operations, and contains as few nonzero values as possible. Many fundamental matrix operations are greatly simplified by first sparsifying a matrix (see [11]) and the problem has applications in areas such as machine learning [28] and in discovering cycle bases of graphs [19] (which is a more simple subcase). But there seem to be only a small number of heuristics for matrix sparsification (e.g., [6] for example), or algorithms under limiting assumptions ([16] considers matrices that satisfy the Haar condition), but no general approximation algorithms. McCormick [21] established that this problem is NP-Complete.

For these two classic problems, we wish to investigate potentials and limits of approximation algorithms both for the general problems and for their variants under simplifying assumptions. To this end, we will need to consider the well-known vector problems min unsatisfy and exact dictionary representation (elsewhere called the sparse approximation or highly nonlinear approximation problem [30]).

The min unsatisfy problem is an intuitive problem on linear equations. Given a system \( Ax = b \) of linear equations (where \( A \) is an integer \( m \times n \) matrix and \( b \) is an integer \( m \)-vector), the problem is to provide a rational \( n \)-vector \( x \); the measure to be minimized is the number of equations not satisfied by \( Ax \). The term “min unsatisfy” was first coined by Arora et al. [2] in a seminal paper on the hardness of approximation, but they claim that the the NP-Completeness of the problem is implicit in a 1978 paper of Johnson and Preparata [17]. Subsequent papers have improved the hardness of approximation result in [2]; the strongest result is due to Dinur and Safra [9], who demonstrated that it is NP-hard to approximate \( \text{min unsatisfy} \) to within a factor of \( 2^{\log^{1-\epsilon}(n)} \) of optimal. For this problem, Berman and Karpinski [3] gave a randomized \( \frac{m}{c \log m} \)-approximation algorithm (where \( c \) is a constant). We know of no heuristics studied for this problem.

The exact dictionary representation problem is the fundamental problem in sparse approximation theory (see [22]). In this problem, we are given a matrix of dictionary vectors \( D \) and a target vector \( s \), and the task is to find the smallest set \( D' \subset D \) such that a linear combination of the vectors of \( D' \) is equal to \( s \). This problem and its variants have been well studied. According to Temlyakov [29], a variant of this problem may be found as early as 1907, in a paper of Schmidt [26]. The problem was shown to be NP-Complete by Natarajan [23]. (See [20] for a discussion of the problem.)

Recently, the field of sparse approximation theory has become exceedingly popular: For example, SPAR05 was largely devoted to it, as was the SparseLand 2006 workshop at Princeton, and also a mini-symposium.
at NYU’s Courant Institute in 2007. The applications of sparse approximation theory are vast and include signal representation [7], amplitude optimization [27] and function approximation [23]. When the dictionary vectors are Fourier coefficients, then this problem is a classic problem in Fourier analysis, with applications in data compression, feature extraction, locating approximate periods and similar data mining problems [34, 13, 14]. There is in fact a host of results for this problem, though all are heuristics or approximations under some qualifying assumptions. In fact, Amaldi and Kann [1] showed that this problem (which they called RVLS – ‘relevant variables in the linear system’) is as hard to approximate as \text{min unsatisfy}, although their result seems to have escaped the notice of the sparse approximation theory community.

**Our contribution.** As a first step, we demonstrate that the matrix problems \text{sparse null space} and \text{matrix sparsification} are equivalent, and that the vector problems \text{min unsatisfy} and \text{exact dictionary representation} are equivalent as well. (Note that although these equivalences are straightforward, [5] claimed that the \text{sparse null space} problem is computationally more difficult than \text{matrix sparsification}.)

We then proceed to show that \text{matrix sparsification} is hard to approximate, via a reduction from \text{min unsatisfy}. It follows that all four problems considered here are hard to approximate within a factor $2^{\log^{1-o(1)} n}$ of optimal.

This hardness result for \text{matrix sparsification} is important in its own right, but it further leads us to ask what can be done for this problem. Specifically, what restrictions or simplifying assumptions may be made upon the input matrix to make \text{matrix sparsification} problem tractable? In addressing this question, we provide the major contribution of this paper and show how to adapt the vast number of heuristics and algorithms for \text{exact dictionary representation} to solve \text{matrix sparsification} (and hence \text{sparse null space} as well). This allows us to conclude, for example, that \text{matrix sparsification} admits a randomized $\frac{m}{c \log m}$ approximation algorithm, and also to give limiting conditions under which a known $\ell_1$ relaxation scheme for \text{exact dictionary matching} solves \text{matrix sparsification} exactly.

Our results also carry over to relaxed version of these problems, where the input is extended by an error term $\lambda$ which relaxes a constraint. We omit details in this paper.

An outline of our paper follows: In Section 2 we review some linear algebra and introduce notation. In Section 3 we prove equivalences between the two matrix problems and the two vector problems. In Section 4 we prove that \text{matrix sparsification} is hard to approximate, and in Section 5 we show how to adapt algorithms for \text{exact dictionary representation} to solve \text{matrix sparsification}.

## 2 Preliminaries

In this section we review some linear algebra, introduce notation and definitions, and formally state our four problems.

### 2.1 Linear algebra and notation.

**Matrix and vector properties.** Given a set $V$ of $n$ $m$-dimensional column vectors, an $m$-vector $v \notin V$ is independent of the vectors of $V$ if there is no linear combination of vectors in $V$ that equals $v$. A set of vectors is independent if each vector in the set is independent of the rest.

Now let the vectors of $V$ be arranged as columns of an $m \times n$ matrix $A$; we refer to a column of $A$ as $a_i$, and to a position in $A$ as $a_{ij}$. We define $\#\text{col}(A)$ to be the number of columns of $A$. The column span of $A$ ($\text{col}(A)$) is the (infinite) set of column vectors that can be produced by a linear combination of the columns of $A$. The column rank of $A$ is the dimension of the column space of $A$ ($\text{rank}(A) = \dim(\text{col}(A))$);
it is the size of the maximal independent subset in the columns of $A$. If the column rank of $A$ is equal to $n$, then the columns of $A$ are independent, and $A$ is said to be full rank.

Other matrices may be produced from $A$ using elementary column operations. These include multiplying columns by a nonzero factor, interchanging columns, and adding a multiple of one column to another. These operations produce a matrix $A'$ which has the same column span as $A$; we say $A$ and $A'$ are column equivalent. It can be shown that $A$, $A'$ are column equivalent iff $A' = AX$ for some invertible matrix $X$.

Let $R$ be a set of rows of $A$, and $C$ be a set of columns. $A(R,C)$ is the submatrix of $A$ restricted to $R$ and $C$. Let $A(:,C)$ ($A(R,:)$) be the submatrix of $A$ restricted to all rows of $A$ and to columns in $C$ (restricted to the rows of $R$ and all columns in $A$). A square matrix is an $m \times m$ matrix. A square matrix is nonsingular if it is invertible.

**Null space.** The null space (or kernel) of $A$ (null($A$)) is the set of all nonzero $m$-length vectors $b$ for which $Ab = 0$. The rank of $A$’s null space is called the corank of $A$. The rank-nullity theorem states that for any matrix $A$, rank($A$) + corank($A$) = $n$. Let $N$ be a matrix consisting of column vectors in the null space of $A$; we have that $AN = 0$. If the rank of $N$ is equal to the corank of $A$ then $N$ is a full null matrix for $A$.

Given matrix $A$, a full null matrix for $A$ can be constructed in polynomial time. Similarly, given a full rank matrix $N$, polynomial time is required to construct a matrix $A$ for which $N$ is a full null matrix [24].

**Notation.** Throughout this paper, we will be interested in the number of zero and nonzero entries in a matrix $A$. Let $\text{nnz}(A)$ denote the number of nonzero entries in $A$. For a vector $x$, let $||x||_0$ denote the number of nonzero entries in $x$. This notation refers to the quasi-norm $\ell_0$, which is not a true norm since $\lambda ||x||_0 \neq ||\lambda x||_0$, although it does honor the triangle inequality.

For vector $x$, let $x_i$ be the value of the $i$th position in $x$. The support of $x$ ($\text{supp}(x)$) is the set of indices in $x$ which correspond to nonzero values, $i \in \text{supp}(x) \iff x_i \neq 0$.

The notation $A|B$ indicates that matrix $B$ is appended to matrix $A$. $M = A \times B$ means that $M$ is formed by multiplying each individual entry in $A$ by the entire matrix $B$. (If $A$ is $m \times n$, $B$ is $p \times q$, then $M$ is $mp \times nq$.)

### 2.2 Approximation equivalence

Here we define approximation equivalence. Some of our notation and definitions are inspired by [1], which itself built upon [18].

**Definition 1** An optimization problem is a four-tuple $F = \{I_F, S_F, M_F, \text{opt}_F\}$, where $I_F$ is the set of input instances, $S_F(x)$ is the solution space for $x \in I_F$, $M_F(x,y)$ is the objective metric for $x \in I_F$ and $y \in S_F(x)$, and $\text{opt}_F \in \{\min,\max\}$.

We will assume throughout the paper that $\text{opt}_F = \min$.

For any optimization problem $F$ and $x \in I_F$, we define $F(x) = \min_{y \in S_F(x)} M_F(x,y)$ and $||F(x)|| = M_F(x,F(x))$. An approximation $\tilde{F}$ to $F$ is any map on $I_F$ with $\tilde{F}(x) \in S_F(x)$. We write $||\tilde{F}(x)||$ for $M_F(x,\tilde{F}(x))$. $\tilde{F}$ is an $\lambda$-approximation for $F$ when, for all $x \in I_F$, $||\tilde{F}(x)|| \leq \lambda||x||$.

**Definition 2** Given optimization problems $F$ and $G$, an exact reduction from $F$ to $G$ is a pair $(t_1,t_2)$ that satisfies the following: (1) $t_1, t_2 \in P$. (2) $t_1 : I_F \to I_G$ and for all $x \in I_F$, $y = t_1(x) \in S_G(t_1(x))$ and $t_2(x,y) \in S_F(x)$. (3) For all $x \in I_F$, $y' \in S_G(t_1(x))$ and $M_F(x,t_2(x,y')) = M_G(t_1(x),y')$. (4) For all $x \in I_F$, $||F(x)|| \geq ||G(t_1(x))||$.

We write $F \leq G$.  

3
Theorem 3 If $F \preceq G$ and $G$ admits a $\lambda$-approximation, then so does $F$.

Proof. We are given that for $G$, there exists $\tilde{G}$ with $||\tilde{G}(x)|| \leq \lambda$ for all $x \in \mathcal{I}_G$. Let $\tilde{F}(x) = t_2(x, \tilde{G}(t_1(x)))$, where $(t_1, t_2)$ is the $F \preceq G$ exact reduction. It suffices to show that $||\tilde{F}(x)|| \leq ||\tilde{G}(x')||$ where $x' = t_1(x)$. By the fourth item of the definition, it suffices to demonstrate that $||\tilde{F}(x)|| \leq ||\tilde{G}(x')||$. By the third item, $||\tilde{F}(x)|| = M_F(x, t_2(x, G(x'))) = M_G(x', G(x')) = ||G(x')||$. □

In fact, it can be shown that $||\tilde{F}(x)|| = ||\tilde{G}(x')||$. If $F \preceq G$ and $G \preceq F$, we say that $F$ and $G$ are equivalent and write $F \sim G$.

Corollary 4 If $F \sim G$, then $F$ admits a $\lambda$-approximation if and only if $G$ admits a $\lambda$-approximation.

2.3 Minimization problems

In this section, we formally state the four major minimization problems discussed in this paper. The first two problems have vector solutions, and the second two problems have matrix solutions.

exact dictionary representation (EDR)

$\mathcal{I}_{EDR} = (D, s)$, $m \times n$ matrix $D$, vector $s$ with $s \in \text{col}(D)$

$S_{EDR}(D, s) = \{v \in \mathbb{R}^n : Dv = s\}$

$m_{EDR}((D, s), v) = ||v||_0$

min unsatisfy (MU)

$\mathcal{I}_{MU} = (A, y)$, $m \times n$ matrix $A$, vector $y \in \mathbb{R}^m$

$S_{MU}(A, y) = \{x : x \in \mathbb{R}^n\}$

$m_{MU}((A, y), x) = ||y - Ax||_0$

sparse null space (SNS)

$\mathcal{I}_{SNS} = \text{matrix } A$

$S_{SNS}(A) = \{N : N \text{ is a full null matrix for } A\}$

$m_{SNS}(A, N) = \text{nnz}(N)$

matrix sparsification (MS)

$\mathcal{I}_{MS} = \text{full rank } m \times n \text{ matrix } B$

$S_{MS}(B) = \{\text{matrix } N : N = BX \text{ for some invertible matrix } X\}$

$m_{MS}(B, N) = \text{nnz}(N)$

3 Equivalences

In this section, we show that min unsatisfy and exact dictionary representation are equivalent. Then we show that exact dictionary representation and min unsatisfy are equivalent.

3.1 Equivalence of vector problems

In this section we show that EDR and min unsatisfy are equivalent.

We reduce EDR to min unsatisfy. Given input $(D, s)$ to EDR, we seek a vector $v$ with minimum $||v||_0$ that satisfies $Dv = s$. Let $y$ be any vector that satisfies $Dy = s$, and $A$ be a full null matrix for $D$. (These can be derived in polynomial time.) Let $x = MU(A, y)$ and $v = y - Ax$. We claim that $v$ is a solution to EDR.
First note that \( v \) satisfies \( Dv = s : Dv = D(y - Ax) = Dy - DAx = s - 0 = s \). Now, the call to \( MU(A, y) \) returned a vector \( x \) for which \( \|y - Ax\|_0 = \|v\|_0 \) is the minimization measure; and, as \( x \) ranges over \( \mathbb{R}^n \), the vector \( v = y - Ax \) ranges over all vectors with \( Dv = s \). Hence, the oracle for \( \min \mathrm{unsatisfy} \) directly minimizes \( \|v\|_0 \), and so \( v \) is a solution to \( \mathrm{EDR} \).

We now reduce \( \min \mathrm{unsatisfy} \) to \( \mathrm{EDR} \). Given input \( \langle A, y \rangle \) to \( \min \mathrm{unsatisfy} \), we seek a vector \( x \) which minimizes \( \|y - Ax\|_0 \). We may assume that \( A \) is full rank. (Otherwise, we can simply take any matrix \( \tilde{A} \) whose columns form a basis of \( \text{col}(A) \), and it follows easily that \( \|MU(A, y)\| = \|MU(\tilde{A}, y)\| \).) Find (in polynomial time) a matrix \( D \) such that \( A \) is a full null matrix for \( D \) (this can be achieved by finding \( D^T \) as a null matrix of \( A^T \)). Let \( s = Dy \), and \( v = \mathrm{EDR}(D, s) \). Since \( Dv = s \) we have that \( D(y - v) = Dy - Dv = 0 \), from which we conclude that \( y - v \) is in the null space of \( D \), and therefore in the column space of \( A \). It follows that we can find an \( x \) such that \( Ax = y - v \). We claim that \( x \) solves the instance of \( \min \mathrm{unsatisfy} \): It suffices to note that the call to \( \mathrm{EDR}(D, s) \) minimizes \( \|v\|_0 = \|y - Ax\|_0 \), and that as \( v \) ranges over \( \{v : Dv = s\} \), the vector \( Ax = y - v \) ranges over all of \( \text{col}(A) \). In conclusion,

**Lemma 5** exact dictionary representation and \( \min \mathrm{unsatisfy} \) are equivalent problems.

### 3.2 Equivalence of matrix problems

In this section we demonstrate that sparse null space and matrix sparsification are equivalent. Recall that in the description of matrix sparsification on input matrix \( B \), we required that \( B \) be full rank, \( \#\text{col}(B) = \text{rank}(B) \). (We could in fact allow \( \#\text{col}(B) > \text{rank}(B) \), but this would trivially result in \( \#\text{col}(B) - \text{rank}(B) \) zero columns in the solution, and these columns are not interesting.) We will need the following lemma:

**Lemma 6** Let \( B \) be a full null matrix for \( m \times n \) matrix \( A \). The following statements are equivalent: (1) \( N = BX \) for some invertible matrix \( X \). (2) \( N \) is a full null matrix for \( A \).

**Proof.** In both cases, \( N \) and \( B \) must have the same number of columns, the same rank, and the same span. This is all that is required to demonstrate either direction. \( \square \)

We can now prove that sparse null space and matrix sparsification are equivalent. sparse null space may be solved utilizing an oracle for matrix sparsification. Given input \( A \) to sparse null space, create (in polynomial time) a matrix \( B \) which is a full null matrix for \( A \), and let \( N = MS(B) \). We claim that \( N \) is a solution to SNS(\( A \)). Since \( N = BX \) for some invertible matrix \( X \), by Lemma 6 \( N \) is a full null matrix for \( A \). Therefore the call to \( MS(B) \) is equivalent to a call to \( MS(N) \), which solves sparse null space on \( A \).

We show that matrix sparsification can be solved using an oracle for sparse null space. Given input \( B \) to matrix sparsification, create (in polynomial time) matrix \( A \) such that \( B \) is a full null matrix for \( A \). Let \( N = \text{SNS}(A) \). We claim that \( N \) is a solution to \( MS(B) \). By the lemma, \( N = BX \) for some invertible matrix \( X \), so \( N \) can be derived from \( B \) via elementary row reductions. The call to \( \text{SNS}(A) \) finds an optimally sparse \( N \), which is equivalent to solving to \( min \text{unsatisfy} \) on \( B \). In conclusion,

**Lemma 7** matrix sparsification and sparse null space are equivalent problems.

### 4 Hardness of approximation for matrix problems

In this section, we prove the hardness of approximation of matrix sparsification (and therefore sparse null space). This motivates the search for heuristics or algorithms under simplifying assumptions for matrix sparsification. For the reduction, we will need a relatively dense matrix which we know cannot be further sparsified. We will prove the existence of such a matrix in the first subsection.
4.1 Unsparsifiable matrices

Any \(m \times n\) matrix \(A\) may be column reduced to contain at most \((m - r + 1)r\) nonzeros, where \(r = \text{rank}(A)\). For example, Gaussian elimination on the columns of the matrix will accomplish this sparsification. We will say that a rank \(r\), \(m \times n\) matrix \(A\) is completely unsparsifiable if and only if, for any invertible matrix \(X\), \(\text{nnz}(AX) \geq (m - r + 1)r\). A matrix \(A\) is optimally sparse if, for any invertible \(X\), \(\text{nnz}(AX) \geq \text{nnz}(A)\).

The main result of this section follows.

**Theorem 8** Let \(A\) be an \(m \times n\) matrix with \(m \geq n\). If every square submatrix of \(A\) is nonsingular, then \(A\) has rank \(n\) and is completely unsparsifiable. Moreover, in such case the matrix \(\binom{I}{A}\) is optimally sparse, where \(I\) is the \(n \times n\) identity matrix.

Before attempting a proof of the theorem, we need a few intermediate results.

**Lemma 9** Matrix \(A\) is optimally sparse if and only if, for any vector \(x \neq 0\), \(\|Ax\|_0 \geq \max_{i \in \text{supp}(x)} |a_i|_0\).

**Proof.** Suppose that there exists an \(x\) that, for some \(i \in \text{supp}(x)\), \(\|Ax\|_0 < |a_i|_0\). Then we may replace the matrix column \(a_i\) by \(Ax\), and create a matrix with the same rank as \(A\) which is sparser than \(A\); a contradiction. Similarly, suppose that \(A\) is not optimally sparse, so that there exists \(B = AX\) with \(\text{nnz}(B) < \text{nnz}(A)\), for some invertible \(X\). Assume without loss of generality that the diagonal of \(X\) is full, \(x_{ii} \neq 0\) (otherwise just permute the columns of \(X\) to make it so). Then there must exist an index \(i \in [n]\) with \(\|b_i\|_0 < |a_i|_0\), and we have \(\|Ax_i\|_0 = \|b_i\|_0 < |a_i|_0 \leq \max_{i \in \text{supp}(x)} |a_i|_0\), since \(x_{ii} \neq 0\). \(\square\)

A submatrix \(A(R, C)\) is row-inclusive if and only if for any row index \(r\), \(A(r, C)\) is in the row span of \(A(R, C)\). Note that this implies that \(A(R, C)\) includes all the rows of \(A(:, C)\) in its row span. A submatrix \(A(R, C)\) is a candidate submatrix of \(A\) (written \(A(R, C) \triangleleft A\)) if and only if \(A(R, C)\) is both row-inclusive and \(\text{rank}(A(R, C)) = |C| - 1\). This last property is equivalent to stating that the columns of \(A(R, C)\) form a circuit – they are minimally linearly dependent. We can potentially zero out \(|R|\) entries of \(A\) by using the column dependency of \(A(R, C)\); being row-inclusive means there would be exactly \(|R|\) zeros in the modified column of \(A\).

The next lemma demonstrates the close relationship between candidate submatrices and vectors \(x\) which may sparsify \(A\) as in Lemma 9.

**Lemma 10** For any \(m \times n\) matrix \(A\): (1) For any \(x \neq 0\) and \(i \in \text{supp}(x)\), there exists \(A(R, C) \triangleleft A\) for which \(|R| \geq m - \|Ax\|_0\), and \(i \in C \subset \text{supp}(x)\). (2) For any \(A(R, C) \triangleleft A\) there exists a vector \(x\) for which \(\text{supp}(x) = C\) and \(\|Ax\|_0 = m - |R|\).

**Proof.** Part 1: Let \(R' = [m] - \text{supp}(Ax)\) (where \([m] = 1, 2, \ldots, m\)), and choose \(C\) so that \(i \in C \subset \text{supp}(x)\), and the columns of \(A(R', C)\) form a circuit. (Note that the columns \(A(R', \text{supp}(x))\) are dependent since \(A(R', :)x = 0\). Now expand \(R'\) to \(R\) so that \(A(R, C)\) is row-inclusive. Then \(\text{rank}(A(R, C)) = \text{rank}(A(R', C)) = |C| - 1\), so that \(A(R, C) \triangleleft A\).

Part 2: Since the columns of \(A(R, C)\) are dependent, there is an \(\hat{x}\) with \(A(R, C)\hat{x} = 0\). Then \(\dim(\text{col}(A(R, C)^T)) = |C| - 1 = \dim(\text{null}(\hat{x}^T))\) and also \(\text{col}(A(R, C)^T) \subset \text{null}(\hat{x}^T)\), which together imply \(\text{col}(A(R, C)^T) = \text{null}(\hat{x}^T)\). So \(A(r, C)\hat{x} = 0\) is true iff \(r \in R\) (using the fact that \(A(R, C)\) is row-inclusive). Now choose \(x\) so that \(x(C) = \hat{x}\) and all other coordinates are zero; then \(\text{supp}(Ax) = [m] - R\). \(\square\)

The following is an immediate consequence of the lemma, and is crucial to our proof of Theorem 8.
Corollary 11 The \( m \times n \) matrix \( A \) is optimally sparse if and only if there is no candidate submatrix \( A(R, C) \subseteq A \) with \( m - |R| < |a_i| \) for some \( i \in C \).

We are now ready to prove the theorem.

Proof of Theorem 8. Let \( B = \left( \frac{i}{j} \right) \). We prove that \( B \) is optimally sparse. Suppose \( B(R, C) \not\subset B \). Let \( R_I = R \cap [n] \) and \( R_A = R - [n] \). Now \( B(R_I, C) \) is a submatrix of \( I \) with dependent columns, so \( B(R_I, C) = 0 \). By row-inclusiveness, \( R_I \) must include all zero rows in \( B(:, C) \), so \( |R_I| = n - |C| \). Since \( B(R_I, C) = 0 \), it follows that \( \text{rank}(B(R_A, C)) = \text{rank}(B(R, C)) = |C| - 1 \), and \( |R_A| \geq |C| - 1 \). Any \( |C| \times |C| \) submatrix of \( B(R_A, C) \) would make the rank at least \( |C| \), so we must have \( |R_A| < |C| \); thus \( |R_A| = |C| - 1 \). Combined with \( |R_I| = n - |C| \), this implies that \( |R| = n - 1 \). Then \( m + n - |R| = m + 1 = |b_i|_0 \) for any column \( b_i \) of \( B \), proving that \( B \) is optimally sparse by corollary 11.

Recall that Gaussian elimination on matrix \( A \rightarrow G \) yields \( \text{nnz}(G) = (m - n + 1)n \). Now suppose there is an invertible matrix \( X \) with \( \text{nnz}(AX) < (m - n + 1)n \). Then \( \text{nnz}(BX) = \text{nnz}(XAX) < n^2 + (m - n + 1)n = (m + 1)n \), contradicting the optimal sparsity of \( B \). Hence no such \( X \) exists and \( A \) is completely unsatisfiable. □

4.2 Reduction for matrix problems

After proving the existence of an unsatisfiable matrix in the last section, we can now prove the hardness of approximation of matrix sparsification. We reduce \text{min unsatisfy} to matrix sparsification. Given an instance \( (A, y) \) of \text{min unsatisfy}, we create a matrix \( M \) such that matrix sparsification on \( M \) solves the instance of \text{min unsatisfy}.

Before describing the reduction, we outline the intuition behind it. We wish to create a matrix \( M \) with many copies of \( y \) and some copies of \( A \). The number of copies of \( y \) should greatly outnumber the number of copies of \( A \). The desired approximation bounds will be achieved by guaranteeing that \( M \) is composed mostly of zero entries and of copies of \( y \). It follows that minimizing the number of nonzero entries in the matrix (solving matrix sparsification) will reduce to minimizing the number of nonzero entries in the copies of \( y \) by finding a sparse linear combination of \( y \) with some other dictionary vectors (solving \text{min unsatisfy}).

The construction is as follows: Take an unsatisfiable \( (p + q) \times p \) matrix \( I_p \) (where \( I_p \) is a \( p \times p \) identity matrix), and create matrix \( M_l = \left( \frac{I_p}{X} \right) + y = \left( \frac{I_p + y}{X} \right) \) (of size \( (p + q)m \times p \)). Further create matrix \( I_q \times A \) (of size \( qm \times qn \)), and take matrix \( 0 \) (of size \( pm \times qn \)) and form matrix \( M_r = \left( \frac{I_q \times A}{0} \right) \) (of size \( (p + q)m \times qn \)). Append \( M_r \) to the right of \( M_l \) to create matrix \( M = M_l | M_r \) of size \( (p + q)m \times (p + qn) \). By a slight abuse of notation, we could summarize this construction as \( M = \left( \begin{array}{cc} I_p & 0 \\ X & I_q \end{array} \right) \left( \begin{array}{c} y \\ A \end{array} \right) \).

In \( M \) there are \( p + pm \) vector entries corresponding to copies of \( y \) out of a total of \( p + pn + qn \) vector entries. Through a prudent choice of \( p \) and \( q \), we can ensure that the term \( pn \) is appropriately larger than all other terms. Specifically, the number of copies of \( y \) can be made to outnumber the other vector entries by a multiplicative factor of \( n^c \) for any constant \( c \).

It follows that the number of zeros in \( M \) depends mostly on the number of zeros induced by a linear combination of dictionary vectors that includes \( y \). Because \( M_l \) is unsatisfiable, vectors in those rows will not contribute to sparsifying other vectors in these rows; only vectors in \( M_r \) may sparsify vectors in \( M_l \). It follows that an approximation to matrix sparsification will yield a similar approximation – within a factor of \( 1 + \frac{1}{n^c} \) – to \text{min unsatisfy}. 

7
5 Solving matrix sparsification through min unsatisfy

In the previous section we showed that matrix sparsification is hard to approximate. This motivates the search for heuristics and algorithms under simplifying assumptions for matrix sparsification. In this section we show how to extend algorithms and heuristics for min unsatisfy to apply to matrix sparsification – and hence sparse null space – while preserving approximation guarantees. (Note that this result is distinct from the hardness result; neither one implies the other.)

We first present an algorithm for matrix sparsification which is in essence identical to the one given by Coleman and Pothen [8] for sparse null space. The algorithm assumes the existence of an oracle for a problem we will call the sparsest independent vector problem. The algorithm makes a polynomial number of queries to this oracle, and yields an optimal solution to matrix sparsification.

The sparsest independent vector problem takes input matrices $A$ and $B$, with at least one column of $A$ independent of $B$, and asks for the sparsest vector that is in the span of $A$, but not in the span of $B$. More formally, sparsest independent vector is defined as follows.

\[
\text{SIV}(A, B) = \{ a : a \in \text{col}(A), a \notin \text{col}(B) \}
\]

The following algorithm reduces matrix sparsification on an $m \times n$ input matrix $B$ to making a polynomial number of queries to an oracle for sparsest independent vector:

```
Algorithm Matrix_Sparsification(A)
B ← null
for i = 1 to n:
    b_i = SIV(A, B)
    B ← B | b_i
return B
```

This greedy algorithm sparsifies the matrix $A$ by generating a new matrix $B$ one column at a time. The first column is the sparsest possible, and each subsequent column is the next sparsest. It is decidedly non-obvious why such a greedy algorithm would actually succeed; we refer the reader to [8] where it is proven that greedy algorithms yield an optimal result on matroids such as the set of vectors in $\text{col}(A)$. Our first contribution is in expanding the result of [8] as follows.

**Lemma 12** Let subroutine $\text{SIV}$ in algorithm Matrix_Sparsification be a $\lambda$-approximation oracle for sparse independent vector. Then the algorithm yields a $\lambda$-approximation to matrix sparsification.

**Proof.** Given $m \times n$ matrix $A$, suppose $\hat{C}$ exactly solves $\text{MS}(A)$, and that the columns $\hat{c}_1, \ldots, \hat{c}_n$ of $\hat{C}$ are sorted in increasing order by number of nonzeros. Let $s_i = ||\hat{c}_i||_0$; then $s_1 \leq s_2 \leq \ldots \leq s_n$. As already mentioned, given a true oracle to sparsest independent vector, algorithm Matrix_Sparsification would first discover a column with $s_1$ nonzeros, then a column with $s_2$ nonzeros, etc.

Now suppose algorithm Matrix_Sparsification made calls to a $\lambda$-approximation oracle for sparse independent vector. The first column generated by the algorithm, call it $b_1$, will have at most $\lambda s_1$ nonzeros, since the optimal solution has $s_1$ nonzeros. The second column generated will have at most $\lambda s_2$ nonzeros, since the optimal solution to the call to $\text{SIV}$ has no more than $s_2$ nonzeros: even if $b_1$ is suboptimal, it is true that at least one of $\hat{c}_1$ or $\hat{c}_2$ is an optimal solution to $\text{SIV}(A, b_1)$. 

More generally, the $i^{th}$ column discovered by the algorithm contains no more than $\lambda s_i$ nonzeros, since at least one of $\{\tilde{c}_1, \ldots, \tilde{c}_i\}$ is an optimal solution to the $i^{th}$ query to SIV. Thus we have $\text{nnz}(B) = \sum_i ||b_i||_0 \leq \sum \lambda ||\tilde{c}_i||_0 = \lambda \text{nnz}(\tilde{C})$, and may conclude that the algorithm yields a $\lambda$-approximation to matrix sparsification. □

It follows that in order to utilize the aforementioned algorithm for matrix sparsification, we need some algorithm for sparsest independent vector. This is in itself problematic, as the sparsest independent vector problem is hard to approximate – in fact, we will demonstrate later that sparsest independent vector is as hard to approximate as min unsatisfy. Hence, although we have extended the algorithm of [8] to make use of an approximation oracle for sparsest independent vector, the benefit of this algorithm remains unclear.

To this end, we will show how to solve sparsest independent vector while making queries to an approximate oracle for min unsatisfy. This algorithm preserves the approximation ratio of the oracle. This implies that all algorithms for min unsatisfy immediately carry over to sparsest independent vector, and further that they carry over to matrix sparsification as well. This also implies a useful tool for applying heuristics for min satisfy to the other problems.

sparsest independent vector on input $\langle A, B \rangle$ asks to find the sparsest vector that is dependent on a group of vectors in the span of $A$ that includes at least a single vector independent of $B$. It is not difficult to see that min unsatisfy solves a similar problem: Given a matrix $A$ and target vector $y$, find the sparsest vector that is dependent on a group of vectors in the span of $A$ that includes $y$. Hence, if we query the oracle for min unsatisfy once for each vector $a_i \notin \text{col}(B)$, one of these queries must return the solution for the sparsest independent vector problem. This discussion implies the following algorithm:

Algorithm SparseIndependentVector($A, B$)

$C, c \leftarrow \text{null}$

$s = n+1$

for $i = 1$ to $n$:

if $a_i \notin \text{col}(B)$

$C \leftarrow A\backslash\{a_i\}$

$x \leftarrow \text{MU}(C, a_i)$

$c' \leftarrow Cx - a_i$

if $||c'||_0 < s$

$c \leftarrow c'$

$s = ||c||_0$

return $c$

Note that when this algorithm is given a $\lambda$-approximate oracle for min unsatisfy, it yields a $\lambda$-approximate algorithm for sparsest independent vector.

We conclude this section by giving hardness results for sparsest independent vector by reduction from min unsatisfy; we show that any instance $\langle A, b \rangle$ of min unsatisfy may be modeled as an instance $\langle A', B' \rangle$ of sparsest independent vector: Let $A' = A|y$, and $B' = A$. This suffices to force the linear combination to include $y$. It follows that sparsest independent vector is as hard to approximate as min unsatisfy, and in fact that the two problems are approximation equivalent.

5.1 Approximation algorithms

We have presented a tool for extending algorithms and heuristics for exact dictionary representation to min unsatisfy and then directly to the matrix problems. When these algorithms make assumptions on the dictionary of EDR, it is necessary to investigate how these assumptions carry over to the other problems.
To this end, we consider here one of the most popular heuristic for EDR \(\ell_1\)-minimization – and the case where it is guaranteed to provide the optimal result. The heuristic is to find a vector \(v\) that satisfies \(Dv = s\), while minimizing \(||v||_1\) instead of \(||v||_0\). (See [32, 31, 10] for more details.) In [12], Fuchs shows that under the following relatively simple condition \(\ell_1\)-minimization provides the optimal answer to EDR.

In the following, we write \(\text{sgn}(x)\) to indicate \(\frac{x}{|x|}\), or zero if \(x = 0\). Given a matrix \(D\) whose columns are divided into two submatrices \(D_0\) and \(D_1\), we may write \(D = (D_0 D_1)\), even though \(D_0\) and \(D_1\) may not be contiguous portions of the full matrix. (The reader may view this as permuting the columns of \(D\) before splitting into \(D_0\) and \(D_1\).)

**Theorem 13 (Fuchs)** Suppose that \(s = Dv\), and that \(||v||_0\) is minimal (so that this \(v\) solves \(\text{EDR}(D, s)\)). Split \(D = (D_0 D_1)\) so that \(D_0\) contains all the columns in the support of \(v\). Accordingly, we split the vector \(v = (v_0^T v_1^T)^T\), in which all coordinates of \(v_0\) are nonzero.

If there exists a vector \(h\) so that \(D_0^T h = \text{sgn}(v_0)\), and \(||D_1^T h||_\infty < 1\), then \(||v||_1 < ||w||_1\) for all vectors \(w \neq v\) with \(Dw = s\).

We extend this result to each of our major problems. The proof of the theorem can be found in the Appendix.

**Theorem 14** \(\text{min unsatisfy}\). Suppose, for a given \(A, y\) pair, that \(x\) minimizes \(||y - Ax||_0\). Split \(y = (y_0^T y_1^T)^T\), and \(A = (A_0^T A_1^T)\) so that \(A_0\) is maximal such that \(y_1 = A_1x\), and let \(v = y_0 - A_0x\). If there is a matrix \(u\) with \(||u||_\infty < 1\) and \(A^T_1 u = -A^T_0 \text{sgn}(v)\), then our reduction of \(\text{MU}(A, y)\) to an \(\ell_1\) approximation of \(\text{EDR}(D, s)\) gives the truly optimal answer.

**matrix sparsification.** For a given \(m \times n\) matrix \(B\), suppose \(C\) minimizes \(\text{nnz}(C)\) such that \(C = BX\) for invertible \(X\). For any \(i \in [n]\), split column \(c_i = (c_{i,0}^T c_{i,1}^T)^T\) so that \(c_{i,0}\) is completely nonzero, and, respectively, \(B = (B_{i,0}^T B_{i,1}^T)\), so that \(c_{i,0} = B_{i,0}x_i\). If, for all \(i \in [n]\), there exists vector \(u_i\) with \(||u_i||_\infty < 1\) and \(B_{i,1}^T u_i = -B_{i,0}^T \text{sgn}(c_{i,0})\), then our reduction algorithm to an \(\ell_1\) approximation of \(\text{EDR}\) via \(\text{min unsatisfy}\) will give a truly optimal answer to this MS instance.

**sparse null space** For a given \(m \times n\) matrix \(V\) with \(\text{rank} c\), suppose matrix \(V\) solves \(\text{SNS}(A)\). For each \(i \in [c]\), split column \(v_i = (v_{i,0}^T v_{i,1}^T)^T\) so that \(v_{i,0}\) is completely nonzero and, respectively, \(A = (A_{i,0}^T A_{i,1}^T)\) so that \(A_{i,0}v_{i,0} = 0\). If, for all \(i \in [c]\), there exists vector \(h_i\) with \(||A^T_{i,1} h_i||_\infty < 1\) and \(A^T_{i,0} h_i = \text{sgn}(v_{i,0})\), then our reduction to an \(\ell_1\) approximation of \(\text{EDR}\) via matrix sparsification and \(\text{min unsatisfy}\) gives a truly optimal answer to this SNS instance.

The following more intuitive conditions give us more insight into which matrices are conducive to \(\ell_1\) approximations. We use \(A^+\) to denote \((A^T A)^{-1} A^T\), the pseudoinverse of \(A\).

**Corollary 15** \(\text{min unsatisfy}\). Suppose matrix \(A = (A_0^T A_1^T)\) is split by an optimal answer as in theorem 14. If \(\text{row}(A_0) \subset \text{row}(A_1)\) and \(||(A^T_1)^+ A^T_0||_{1,1} < 1\), then our \(\ell_1\) approximation scheme will give a truly optimal answer.

**matrix sparsification.** Suppose matrix \(B = (B_{i,0}^T B_{i,1}^T)\) is split by the columns of an optimal answer \(C = BX\) as in theorem 14. If, for any \(i\), \(\text{row}(B_{i,0}) \subset \text{row}(B_{i,1})\) and \(||(B^T_{i,1})^+ B^T_{i,0}||_{1,1} < 1\), then our \(\ell_1\) approximation scheme will give a truly optimal answer.

**sparse null space.** Suppose matrix \(A = (A_{i,0}^T A_{i,1}^T)\) is split by the columns of an optimal answer \(V\) with \(AV = 0\) as in theorem 14. If, for any \(i\), \(\text{col}(A_{i,0}) \subset \text{col}(A_{i,1})\) and \(||A^T_{i,0} A_{i,1}||_{1,1} < 1\), then our \(\ell_1\) approximation scheme will give a truly optimal answer.
References

[1] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical computer science*, 147(1–2):181–210, 1995.

[2] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and linear equations. *JCSS*, 54(2), 1997.

[3] Piotr Berman and Marek Karpinski. Approximating minimum unsatisfiability of linear equations. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(25), 2001.

[4] M. Berry, M. Heath, I. Kaneko, M. Lawo, R. Plemmons, and R. Ward. An algorithm to compute a sparse basis of the null space. *Numer. Math.*, 47:483–504, 1985.

[5] R.A. Brualdi, S. Friedland, and A. Pothen. The sparse basis problem and multilinear algebra. *SIAM Journal on Matrix Analysis and Applications*, 16(1):1–20, 1995.

[6] S. Frank Chang and S. Thomas McCormick. A hierarchical algorithm for making sparse matrices sparser. *Mathematical Programming*, 56(1–3):1–30, August 1992.

[7] R. Coifman and M. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38(2):713–718, 1992.

[8] T.F. Coleman and A. Pothen. The null space problem I. complexity. *SIAM Journal on Algebraic and Discrete Methods*, 7(4):527–537, October 1986.

[9] Irit Dinur and Shmuel Safra. On the hardness of approximating label cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(15), 1999.

[10] David L. Donoho. For most large underdetermined systems of linear equations, the minimal l1-norm solution is also the sparsest. http://www-stat.stanford.edu/~donoho/Reports/2004/11l0EquivCorrected.pdf, 2004.

[11] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[12] J.J. Fuchs. On sparse representations in arbitrary redundant bases. *IEEE Trans. Inf. Th.*, 50(6):1341–1344, June 2004.

[13] A.C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. In *STOC*, 2002.

[14] A.C. Gilbert, S. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal sparse fourier representations. In *Proc. SPIE Wavelets XI*, 2005.

[15] J.R. Gilbert and M.T. Heath. Computing a sparse basis for the null space. *SIAM Journal on Algebraic and Discrete Methods*, 8(3):446–459, July 1987.

[16] A.J. Hoffman and S.T. McCormick. A fast algorithm that makes matrices optimally sparse. In *Progress in Combinatorial Optimization*. Academic Press, 1984.

[17] D.S. Johnson and F.P. Preparata. The densenst hemisphere problem. *Theoret. Comput. Sci.*, 6:93–107, 1978.

[18] Viggo Kann. Polynomially bounded minimization problems that are hard to approximate. *Nordic Journal of Computing*, 1(3):317–331, Fall 1994.
[19] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. A faster algorithm for minimum cycle basis of graphs. In Automata, Languages, and Programming: 31st International Colloquium, ICALP 2004 Proceedings, Lecture Notes in Computer Science. Springer, 2004.

[20] J. Matoušek and B. Gartner. Understanding and Using Linear Programming. Springer, 2007.

[21] S.T. McCormick. A combinatorial approach to some sparse matrix problems. PhD thesis, Stanford Univ., Stanford, California, 1983. Technical Report 83-5, Stanford Optimization Lab.

[22] S. Muthukrishnan. Nonuniform sparse approximation with haar wavelet basis. DIMACS TR:2004-42, 2004.

[23] B.K. Natarajan. Sparse approximate solutions to linear systems. SIAM J. on Comput., 24(2):227–234, 1995.

[24] T. Neylon. Sparse solutions for linear prediction problems. PhD thesis, New York University, New York, New York, 2006.

[25] A. Pothen. Sparse null bases and marriage theorems. PhD thesis, Cornell University, Ithica, New York, 1984.

[26] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen, i. Math. Annalen., 64(4):433–476, 1906–1907.

[27] S. Singhal. Amplitude optimization and pitch predictin in multi-pulse coders. IEEE Transactions on Acoustics, Speech and Signal Processing, 37(3), March 1989.

[28] A. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In Proc. 17th International Conf. on Machine Learning, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

[29] V. Temlyakov. Nonlinear methods of approximation. Foundations of Comp. Math., 3(1):33–107, 2003.

[30] J.A. Tropp. Greed is good: algorithmic results for sparse approximation. IEEE Transactions on Information Theory, 50(10):2231–2242, October 2004.

[31] J.A. Tropp. Just relax: convex programming methods for subset selection and sparse approximation. IEEE Transactions on Information Theory, 50(3):1030–1051, March 2006.

[32] J.A. Tropp. Recovery of short, complex linear combinations via $\ell_1$ minimization. IEEE Trans. Inform. Theory, 51(1):188–209, January 2006.

[33] X. Zhao, X. Zhang, T. Neylon, and D. Shasha. Incremental methods for simple problems in time series: algorithms and experiments. In Ninth International Database Engineering and Applications Symposium (IDEAS 2005), pages 3–14, 2005.

[34] J. Zhou, A. Gilbert, M. Strauss, and I. Daubechies. Theoretical and experimental analysis of a randomized algorithm for sparse fourier transform analysis. Journal of Computational physics, 211:572–595, 2006.
6 Appendix

Proof of Theorem 14. min unsatisfy. As in our reduction from MU to EDR, we find matrix D with DA = 0 and vector s = Dy. Then

\[
\left( \begin{array}{c} \text{sgn}(v_0) \\ u \end{array} \right) \in \text{null}(A^T) = \text{col}(D^T) \implies \exists h : D^T h = \left( \begin{array}{c} \text{sgn}(v) \\ u \end{array} \right).
\]

Splitting D = (D_0 D_1), we see that D_0^T h = sgn(v) and ||D_1^T h||_\infty < 1, exactly what is required for theorem 13, showing that \( \ell_1 \) minimization gives the answer D_0 v_0. Since D_0 v_0 = (D_0 D_1) \begin{pmatrix} v_0 \\ 0 \end{pmatrix} = D(y - Ax) = s, this completes the proof.

matrix sparsification. Here, we’ll write A \( \setminus \) i to denote matrix A with the i\textsuperscript{th} column removed. In our reduction of MS to MU, we need to solve instances of MU over equations of the form \((B \setminus i)x = b_i\). According to the MU portion of this theorem, it suffices to show that \((B_{i,1} \setminus i)^T u_i = -(B_{i,0} \setminus i)^T \text{sgn}(c_{i,0})\). The condition for this portion of the theorem implies this, since removing any corresponding rows from a matrix equation of the form Ax = By still preserves the equality.

sparse null space. As in our reduction from SNS to MS, we find a matrix B such that A is a full null matrix for B. For any i, let \( u_i = A^T_{i,1} h_i \) so that \( A^T h_i = \left( \begin{array}{c} \text{sgn}(v_{i,0}) \\ u_i \end{array} \right) \). Then \( \left( \begin{array}{c} \text{sgn}(v_{i,0}) \\ u_i \end{array} \right) \in \text{col}(A^T) = \text{null}(B^T) \), and \( B^T_{i,1} u_i = -B^T_{i,0} \text{sgn}(v_{i,0}) \), which is exactly what is necessary for matrix sparsification to function through \( \ell_1 \) approximation. \( \square \)