Fast Blind MIMO Decoding through Vertex Hopping

Thomas R. Dean, Member, IEEE, Jonathan R. Perlstein, Mary Wootters, Member, IEEE, and Andrea J. Goldsmith, Fellow, IEEE

Abstract—We present an algorithm that efficiently performs blind decoding of MIMO signals. That is, given no channel state information (CSI) at either the transmitter or receiver, our algorithm takes a block of samples and returns an estimate of the underlying data symbols. In prior work, the problem of blind decoding was formulated as a non-convex optimization problem. In this work, we present an algorithm that efficiently solves this non-convex problem in practical settings. This algorithm leverages concepts of linear and mixed-integer linear programming. Empirically, we show that our technique has an error performance close to that of zero-forcing with perfect CSI at the receiver. Initial estimates of the runtime of the algorithm presented in this work suggest that the real-time blind decoding of MIMO signals is possible for even modest-sized MIMO systems.

Index Terms—MIMO, Multiuser detection, Blind source separation, Optimization

I. INTRODUCTION

In this work we propose an efficient method to blindly estimate MIMO channels and decode the underlying transmissions. A previous work, [1], has shown that as long as the channel gain matrix is non-singular, the geometry of the constellation can be exploited to recover the underlying data up to some small amount of remaining ambiguity. In [1], the authors formulate the problem of blind MIMO decoding as a non-convex optimization problem and provide theoretical guarantees as to when the interior point algorithm, with a logarithmic barrier function, correctly solves this problem. As the size of the MIMO system grows, interior-point based algorithms given in [1] become ineffective due to both an increasing proportion of spurious optima as well as numerical instability. In this work, we propose an algorithm inspired by techniques commonly used to solve linear and mixed-integer programs that allow us to perform blind decoding on higher order MIMO systems. More importantly, this approach is far more computationally efficient than the approach in [1], such that real-time decoding is possible.

A blind decoding algorithm that is realistic in terms of both sample size requirements and computational complexity has important applications. Current trends in wireless system design are moving towards systems with shorter wavelengths, higher user mobility and more antennas per user [2]–[4]. Thus, channel state information (CSI) is increasingly rapidly varying and difficult to acquire. Additionally, new systems being proposed demand increased reliability and decreased latency [5]. These factors combined imply that reducing resource overhead from channel estimation will become even more important in the development of future wireless systems.

An efficient blind decoding algorithm can enable high-rate communications in environments where the channel changes too rapidly to be measured or where communication occurs in short bursts. Capacity of channels with CSI unknown at the receiver have been considered extensively in the literature (e.g., [6]–[8] and references therein). While it is theoretically possible to achieve reliable, high-rate communications in these conditions, few schemes exist that do so practically.

A number of previous works have considered blind decoding of MIMO systems, notably [9], [10]. See [1] for a more complete overview of these works. In this work, we compare the performance of our new approach only to [1]. This is because previous approaches have prohibitively large sample size requirements, often growing exponentially in the number of transmit antennas. To our knowledge, [1] is the only blind MIMO decoding method that has sample size requirements that are less than the block length of modern wireless systems.

We now highlight several notable features of the algorithm presented in this work:

- For an $n \times n$ MIMO system we can typically decode a block of $k$ channel uses in $O(n^3k)$ operations. We also characterize the scenarios where the number of operations for decoding exceeds this bound. For comparison, when CSI is known perfectly at the receiver, efficient MIMO decoding algorithms such as zero-forcing and MMSE require $O(n^3k)$ operations to decode $k$ channel uses.
- In the limit of high SNR, given appropriate inputs described in [1], our approach solves the blind decoding problem with a success rate approaching 1 for systems as large as $n = 12$. In comparison, the success rate of the approach in [1] was bounded below 1 beyond $n = 5$ and negligibly small beyond $n = 8$.
- We implement the proposed algorithm in the Rust programming language and show that the runtime of our algorithm is several orders of magnitude faster than the approach given in [1]. For $n \leq 8$ our implementation is approaching fast enough runtimes to enable real-time blind decoding of data streams without specialized
Tables I and II. Runtimes on a single core of a 2.2GHz Intel i7

| n | k | Pr. Success | Time (s) | Pr. Success | Time (s) |
|---|---|-------------|---------|-------------|---------|
| 2 | 8 | 1.00 | 1.83e-3 | 0.99 | 3.01e-2 |
| 3 | 13 | 1.00 | 6.46e-5 | 0.99 | 6.33e-2 |
| 4 | 18 | 1.00 | 1.75e-4 | 0.99 | 0.13 |
| 5 | 18 | 1.00 | 2.50e-4 | 0.97 | 0.30 |
| 6 | 22 | 1.00 | 8.03e-4 | 0.93 | 0.59 |
| 8 | 30 | 0.99 | 3.52e-3 | 0.80 | 3.5 |
| 10 | 100 | 0.99 | 1.91e-2 | 0.75 | - |
| 12 | 144 | 0.99 | 2.28e-1 | 0.75 | - |

In Table I, we report the runtime of an implementation of our algorithm written in the Rust programming language, run on a single core of an Intel i7 2.2GHz processor. These numbers are preliminary as there are many more optimizations remaining to be implemented in our solver. For comparison, we also report the runtime of the gradient descent algorithm of [1], implemented using MATLAB’s `fmincon` solver. In addition, we report that probability that both algorithms return a correct solution to the blind decoding problem. For all values of n, the approach presented in this paper is both faster and more reliable. A more detailed discussion of both the runtime and success probability is presented in Section IV.

The remainder of this paper is organized as follows. Section II describes the system model and notation used throughout this work. Section III revisits several relevant theoretical results presented in [1]. Section IV provides a high-level description of our new algorithm along with empirical results. Sections V and VI describe the core components of our algorithm. Section VII discusses our algorithm in the presence of AWGN. Conclusions are offered in Section VIII.

II. System Model and Notation

In this work we consider $n \times n$ real-valued channel gain matrices, denoted $A$. In our simulations, we draw $A$ where each entry is i.i.d. and normally distributed with zero mean and unit variance, $\mathcal{N}(0, 1)$. Our technique requires only that $A$ be full rank and thus $A$ may be drawn from an arbitrary distribution. We assume AWGN, drawn from $\mathcal{N}(0, \sigma^2)$, is present in the channel. Finally, we consider a block fading model where the channel gain matrix is constant over a period of $k$ channel uses after which it is redrawn independently. This work focuses on the transmission of BPSK signals over such channels.

The vector $x \in \{-1, +1\}^n$ denotes the symbols transmitted in a single channel use and the matrix $X \in \{-1, +1\}^{n \times k}$ denotes the set of symbols transmitted over a single block of $k$ channel uses. Likewise, single observations and blocks of symbols at the receiver are denoted as $y$ and $Y$ respectively.

We assume that no CSI is available at either the transmitter or the receiver. Without the aid of pilot symbols or any knowledge of the underlying data symbols, the receiver attempts to recover an estimate of $X$, denoted $\hat{X}$. However, as discussed in [1], without additional side information, the receiver is only capable of recovering $X$ up to an acceptable transform matrix (ATM), meaning that within each block, $X$ is correct up to permutation and negation of the rows. We denote the set of ATMs as $T$. In the high SNR limit, we say that an algorithm solves the blind decoding problem if, given only $Y$ as input, it returns a value of $X$ that is equivalent to $X$ up to an ATM.

The $i$th column of the matrix $A$ is denoted as $a_i$ and its $j$th row as $a_j$. The vector $e_i$ represents the $i$th element of the standard basis. The set $\text{cols}(X)$ denotes the set of vectors that comprise the columns of $X$ and $\text{vec}(X)$ denotes the $nk \times 1$ vector that consists of the entries of $X$ in row-major order. The notation $U^{-T}$ is shorthand for $(U^{-1})^T$. The symbol $R_+$ denotes the domain of non-negative real numbers, whereas $R_{++}$ denotes the positive real numbers.

III. Fitting a Parallelepiped

In [1], the authors formulate the blind decoding problem as the following non-convex optimization problem:

$$
\begin{align*}
\text{maximize} & \quad \log | \det U | \\
\text{subject to} & \quad \|Uy_i\|_\infty \leq 1 + c \cdot \sigma, \quad i = 1, \ldots, k,
\end{align*}
$$

(1)

where $c$ is some margin which is chosen based on the noise variance. Given proper input, the set of optimal $U$ are equivalent to $A^{-1}$ up to an ATM. Geometrically, this problem can be interpreted as fitting the minimum volume parallelepiped that matches the observed samples. In [1], the authors show that despite the fact that the problem is not convex, under certain assumptions, gradient descent returns the correct solution with high probability. Here, we briefly recount several important theoretical facts proven in [1] about the problem given by (1–2). We initially focus on the noiseless case ($\sigma = 0$), and return to the case $\sigma \neq 0$ in Section VII.

We also assume that $Y$ is full rank; if it is not then (1–2) is not a well-posed problem.

In (2), each $y_i$ imposes two linear constraints on each row $u_{ij}$, that is $-1 \leq (u_{ij}, y_i) \leq 1$, for all $i, j$. The feasible region is thus an $n^2$-dimensional polytope. We say that a given $U$ is at a vertex of this polytope if $UY \in \{-1, +1\}^{n \times k}$. Note that the objective function is not defined at all vertices of the feasible region; if $UY$ is not full rank, this implies $U$

1While we only consider $n \times n$ real channels, we note that the results in this work can be extended to $n \times \infty$ complex-valued channels by considering the usual $2n \times 2n$ equivalent real-valued channel gain matrix and can be extended rectangular channels as discussed in [1]. The results in [1] extend to general MPAM constellations; the performance of the algorithms presented here under the presence of higher-order modulation is a topic of on going research.
is singular and the value of $\mathbf{1}$ is not defined. If two vertices $\mathbf{U}_1$ and $\mathbf{U}_2$ are adjacent (share an edge of the polytope) then this also implies that the Hamming distance between $\mathbf{U}_1\mathbf{Y}$ and $\mathbf{U}_2\mathbf{Y}$ is 1.

A matrix $\mathbf{X} \in [-1, +1]^{n \times k}$, and corresponding set $\text{cols}(\mathbf{X}) \subseteq [-1, +1]^n$ with $k \geq n$, has the maximal subset property (MSP) if there is a subset $\text{cols}(\mathbf{V}) \subseteq \text{cols}(\mathbf{X})$ of size $n$ so that if $\mathbf{V} \in \mathbb{R}^{n \times n}$ is the matrix with elements of $\text{cols}(\mathbf{V})$ as columns, then

$$|\det \mathbf{V}| = \max_{\mathbf{W} \in [-1, +1]^{n \times n}} |\det \mathbf{W}|.$$  

Since $\det(\mathbf{V})$ is linear in the columns of $\mathbf{V}$, then $\mathbf{V}$ also maximizes the determinant amongst all matrices in $\{-1, +1\}^{n \times n}$. We note the following additional facts about the program given in $\mathbf{1}$–$\mathbf{2}$:

- If the matrix $\mathbf{X}$ has the maximal subset property, then the set of global optima of $\mathbf{1}$–$\mathbf{2}$ contain all solutions to the blind decoding problem. This is proven in $\mathbf{1}$.
- The gradient of the objective function is given by

$$\nabla (\log |\det \mathbf{U}|) = \mathbf{U}^{-\top}. \quad (3)$$  

- For $n < 6$, all optima of $\mathbf{1}$–$\mathbf{2}$ are global optima. In $\mathbf{1}$ the authors give specific values of $\mathbf{X}$ that guarantee all optima of $\mathbf{1}$–$\mathbf{2}$ are solutions to the blind decoding problem for $n \leq 4$. The case $n = 5$ is discussed in $\mathbf{13}$.
- Solutions to the blind decoding problem lie on vertices of the feasible region. When $n$ is such that a Hadamard matrix exists, all optima are strict and lie on vertices. In all cases, optima will only lie on the boundary of the feasible region. This is proven in $\mathbf{1}$.

While it is shown in $\mathbf{1}$ that the interior point method with a logarithmic barrier function provably solves the non-convex blind decoding problem for small $n$, this approach has several practical limitations. First, for $n > 5$, local optima exist, and this approach is no longer guaranteed to be correct. As $n$ grows substantially beyond 5, the proportion of optima that are local increases and thus this approach becomes less effective. Second, this approach is not efficient from a computational perspective. All solutions are located on vertices of the feasible region. In general, barrier methods are not well suited to solve this class of problems $\mathbf{14}$. The gradient of the objective function varies rapidly near the boundary of the feasible region which, along with the large number of linear constraints imposed by $\mathbf{2}$, creates numerical instability. As a result, even at low dimension, off-the-shelf solvers such as MATLAB’s fmincon will require a large number of Newton steps to converge. Further, in our numerical experiments in solving $\mathbf{1}$–$\mathbf{2}$, we have observed that such interior point solvers almost never converge for $n > 8$.

**IV. ALGORITHM OVERVIEW AND PERFORMANCE**

The program $\mathbf{1}$–$\mathbf{2}$ is not linear, nor is it even convex. One should not necessarily expect tools from convex optimization, let alone linear programming, to work well. However, we adapt such techniques by leveraging two facts about the problem geometry: the fact that the objective function is multilinear in the rows of $\mathbf{U}$, and that solutions to the blind decoding problem lie on vertices of the feasible region. Informally, our algorithm attempts to find an appropriate solution to the blind decoding problem by hopping between vertices of the feasible region in a similar manner to the simplex algorithm. For this reason, we refer to our algorithm as the *vertex hopping algorithm*. In this section, we give a high-level overview of our approach to blind decoding as well as empirical results demonstrating its BER performance.

The first step of the vertex hopping algorithm is finding a value of $\mathbf{U}$ that is at a non-singular vertex of the feasible region. We refer to this process as Vertex Finding. It turns out that finding such a vertex is a non-trivial task and is often the majority of the work in solving the problem. This procedure is described in Section $\mathbf{V}$.

Once we have found an initial non-singular vertex of the feasible region, we efficiently exploit neighboring vertices of the feasible region in search of a global optimum. This is accomplished in a similar manner as the simplex algorithm. We form a tableau from the linear constraints that define the feasible region and hop between vertices by performing Gauss-Jordan pivots on this data structure. This procedure is described in greater depth in Section $\mathbf{VI}$.

At each step, we hop to the neighboring vertex that has the largest increase in objective function and backtrack if we find a local optimum. We discuss how to identify when we are at a local versus a global optimum in Section $\mathbf{VI}$–$\mathbf{D}$. Local optima are rare and do not exist for $n \leq 5$. In nearly every case the algorithm terminates after at most a few hops; we elaborate on when this is not the case in Section $\mathbf{VI}$.
success rate of our approach when $X$ is drawn uniformly at random for various values of $n$. Our vertex hopping algorithm succeeds with near 100% likelihood when $X$ has the theoretical properties to ensure recovery, given in [1].

A. Numerical Results

In Figure 1 we show empirical results regarding the resulting BER of our scheme in the presence of AWGN. We compare our algorithm to zero-forcing with perfect CSI as well as to the gradient-descent based approach given as Algorithm 1 in [1], which we henceforth refer to as simply ‘gradient descent’. We see that at low SNR, our algorithm nearly matches zero-forcing. At high SNR, our approach yields an increased BER over schemes like zero-forcing. This is caused by the rounding procedure used in our vertex finding algorithm presented in Section VII. We discuss why performance degradation arises in more detail in Section VI-B. It is an open problem to develop alternative approaches for high SNR that mitigate this noise enhancement. We also compare our algorithm to maximum-likelihood decoding with imperfect CSI. We outperform ML decoding with an estimation error modeled as AWGN with variance as low 1% of the channel noise variance.

In Figure 2 we give the success probability of the vertex hopping algorithm for various values of $n$ in the limit of high SNR, where success is defined as properly recovering $X$ correctly up to an ATM. Here $X$ is generated uniformly at random, and the success probability of our algorithm almost exactly matches the empirical probability that a random $X$ has the correct theoretical guarantees provided in [1]. We outperform the gradient descent algorithm in [1] beyond $n = 5$.

B. Performance Analysis

A rigorous theoretical analysis of the time complexity of our algorithm is beyond the scope of this paper. However, we informally comment on the typical performance observed by our algorithm. Our algorithm treats the blind decoding problem as a mixed-integer linear program; optimizing such programs is classic example of an NP-complete problem [15]. Thus, it is likely that exactly solving (1)–(2) becomes hard for large $n$.

We note, however, that in case that occurs with non-negligible probability for $n \leq 4$, the initialization step given in Section VII will directly return a global optima. The time complexity of this step of our algorithm is considered in more depth in Section VII where it is shown to be $O(n^3k)$. Further, we have observed that for $n \leq 8$, if the problem is not solved by the initialization step, we typically require only one or two vertex hops. As described in Section VI-C a single iteration of the pivoting process requires only $O(n^3)$ operations. Thus in practice, our algorithm typically performs near the limit of $O(n^3k)$. Empirically, we find that for $n \leq 5$, the runtime of our algorithm indeed scales roughly linearly with $k$. For larger values of $n$, we discuss the dependence of the runtime on $k$ in Section VI-E.

For $n \geq 6$, where local optima exist, we occasionally encounter a ‘trap’ case as described in Section VI-B. In these cases the solver often enumerates a large subspace before exiting, thus inflating the average runtime. Beginning at $n = 8$, the runtime of our algorithm begins to noticeably increase with $n$. We note that the spectrum of possible determinants of $\pm 1$-valued matrices grows rapidly as $n$ grows (see [16]) and we conjecture that this rapid growth of possible determinants leads to the increased runtime of our algorithm with $n$.

V. Initialization

Linear programs are often solved by the simplex technique which hops between vertices of the feasible region in order to find a global optimum. Typically the simplex technique begins at the origin of the feasible region. When the origin of a linear program is not feasible, techniques exist to find a suitable feasible solution, often termed a basic feasible solution or BFS [17]. Unfortunately, it is not clear how to leverage standard techniques to solve our problem. In our case, the origin is singular, as are a majority of the vertices of the feasible region, and so in particular the gradient is not defined and we cannot start our vertex-hopping technique at these points.

We now present our algorithm for finding a suitable BFS, which is summarized in Algorithm 1 and is referred to as the ‘vertex finding’ process. The $\ell_\infty$ constraints in (2) can be expressed as $2kn \times n^2$ linear inequality constraints: let $U = \text{vec}(\tilde{U})$, then (2) can be expressed as $\tilde{Y}U \leq \mathbb{1}$ for some appropriate $\tilde{Y}$. Suppose $U$ satisfies $l$ constraints with equality. Then we form the $l \times n^2$ matrix $B$ by taking the appropriate rows of $\tilde{Y}$. The matrix $B$ now describes the active constraints. If we consider $\tilde{U} = U + \Delta$ and $\Delta \in \text{null}(B)$, then $\tilde{U}$ will still, at a minimum, satisfy the same $l$ constraints with equality. We refer to the process of setting $\tilde{U} = U + \Delta$ for some $\Delta \in \text{null}(B)$ as “moving within the nullspace of the active constraints”.

We begin by choosing a random feasible point using the technique described in [1]. Our technique successively solves one-dimensional optimization problems. At each step, $U$ is perturbed so that at least one additional constraint becomes
active. This is accomplished by projecting the gradient onto the nullspace of the active constraints. The solver then moves in this direction, leaving the already active constraints unchanged, until an additional constraint becomes active. As a result, when Algorithm 1 terminates, we are guaranteed that at least $n^2$ constraints are active. A graphic depiction of this process is shown in Figure 3.

Algorithm 1 Vertex Finding

Input: An $n \times k$ matrix of received samples $Y$. Any feasible, full rank starting point $U$.

Output: A matrix $U$ that satisfies at least $n^2$ linearly independent constraints of $\mathbb{C}$ with equality.

1: while $B$ is not full rank do
2: $\Delta = U^{-\top}$
3: Find $N$, a basis for null($B$).
4: Compute $\Delta = \text{proj}_N \Delta$.
5: Find max $t \in \mathbb{R}_+$ such that $\| (U + t\Delta)Y \|_\infty = \pm 1$.
6: Update $B$.
7: end while
8: return $U$

Algorithm 1 will return a value of $U$ such that at least $n^2$ entries of $UY$ will be equal to $\pm 1$, with at least $n$ of these entries per row. Because the remaining entries of $X = UY$ are not independent, we often find that most, if not all, entries equal $\pm 1$. In Section V.E we discuss how often Algorithm 1 returns a suitable $X$, and more complete empirical results regarding the distribution of the elements of $UY$ at the output of Algorithm 1 are given in Section VII.

After running Algorithm 1 the resulting columns of $\hat{X}$ must be separated into ‘good’ versus ‘bad’ columns. Good columns are those that lie in $\{-1, +1\}^n$. We let $Y_g$ and $Y_b$ denote the matrices composed of corresponding good and bad columns of $Y$ respectively. If $Y_g$ is not full rank, then we must rerun Algorithm 1 again until a suitable $X$ is obtained. For sufficiently large $k$, it is very rare in the noiseless case that Algorithm 1 fails to produce a suitable output; we further quantify when this happens in Section V.E.

We now briefly comment on the time complexity of Algorithm 1. Algorithm 1 will require $n^2$ iterations to ensure that at least $n^2$ constraints become active. Computing the inverse of $U$ requires $O(n^3)$ operations. The process of projecting the gradient onto the nullspace of the active constraints is considered in Section V.A where it is shown that this requires at most $O(n^3)$ operations per call. Finally, updating $B$ requires computing the product $\Delta Y$ which requires $O(n^3k)$ operations. Since $k > n$, this means that Algorithm 1 requires $O(n^4k)$ operations.

A. Efficient Projection

At each iteration of Algorithm 1 we must project the gradient onto the nullspace of the active constraints. Since the matrix $B$ has $n^2$ rows, a naive approach to finding the nullspace of $B$ would require $O(n^3)$ operations. In this subsection, we describe how to efficiently exploit the sparse structure of the matrix $B$ to efficiently project a vector away from $B$.

We begin by considering the structure of the matrix $B$. At each iteration of the inner loop of Algorithm 1 at least one more constraint becomes active. Suppose that there are $l$ active constraints at the beginning of an iteration of Algorithm 1 and that at the completion of this iteration, $\hat{x}_{ij}$ (where $X = UY$) becomes $\pm 1$. In this case, we append the following row to $B$:

$$b^{(l)} = \left[ 0_{i \times n}, y^{(j)} \right]^T 0_{(n-1-k) \times n} \right].$$

That is, the $l$th row of $B$ will consist of $i \times n$ zeros, followed by the $j$th column of $y$, with zeros in the remaining entries. If multiple constraints become active in a single iteration, we simply add additional rows to $B$ in the same manner. From this discussion, it is clear that the rows of $B$ can be ordered to obtain a block diagonal structure. At most, $B$ will consist of $n$ blocks that are $l \times n$ large. In each iteration of Algorithm 1 we can easily insert each row of $B$ appropriately to maintain this block structure.

Given this block matrix, an efficient procedure to find the component of $\Delta$ that lies in null($B$) is described in Algorithm 2. The algorithm first finds an orthonormal basis for the subspace span$(B)$. The block structure of $B$ means that many of its rows are already orthogonal, this means the matrix $C = BB^T$ is nearly diagonal. Given a basis for span($B$), we obtain the desired output by projecting vec($\Delta$) away from each basis vector.

We now consider the runtime of this procedure. The matrix $C$ need not be updated from scratch at each step and can be computed block-wise. We also note that computing $C$ gives all inner products needed in line 4 of Algorithm 2 and each inner product requires $O(n)$ operations. At most we will need to compute $n^2$ of these inner products per call to Algorithm 2. Similarly, for the loop in lines 13-15, the matrix $B$ will have at most $n^2$ rows and each vector rejection operation will require $O(n)$ operations. At worst, this process will require $O(n^3)$ operations. However, in practice, we find that for most calls to Algorithm 2 each block in $C$ has fewer than $n$ rows and $C$ will only have one or two non-zero entries. Thus we typically perform much better than this bound.
Algorithm 2 Efficient Projection

**Input:** The $l \times n^2$ matrix $B$, the $n \times n$ matrix $\Delta$.

**Output:** The component of $\Delta$ that lies in null($B$).

1. Compute $C = BB^T$. Save indices of non-zero off-diagonal entries of $C$.
2. for each $i < j$ such that $c_{ij} \neq 0$ do
   3. $u = b^{(i)}$, $v = b^{(j)}$
   4. $v = v - (uv^T/uu^T)u$
   5. if $\|v\| > 0$ then
      6. $b^{(j)} = v/\|v\|$
   7. else
      8. Store $j$ as redundant.
3. end if
4. end for
5. Remove redundant rows of $B$.
6. $\delta = \text{vec}(\Delta)$
7. for each row $b^{(i)}$ in $B$ do
6. $\delta = \delta - (\delta \cdot (b^{(i)})^T)b^{(i)}$
8. end for
9. return mat($\delta$)

VI. VERTEX HOPPING

A. Graph of Vertices

We know that solutions of the blind decoding problem will only occur on vertices and thus we attempt to solve the blind decoding problem by searching these vertices. We note that there are at most $2n^2$ possible vertices $UY$ where $U \in \mathbb{R}^{n \times n}$ and $Y \in \mathbb{R}^{n \times k}$. Note that if $k > n$ strictly, then a vertex is determined entirely by any $n$ linearly independent columns of $Y$. Additionally, if $k > n$, then some of the $2n^2$ vertices may become infeasible.

Our search process, summarized in Algorithm 3 begins with a vertex determined by Algorithm 1. Given such a point we can restrict ourselves to performing optimization on a graph that represents the vertices of the feasible region. In this graph, there are at most $2n^2$ nodes corresponding to vertices of the feasible region as described above. An edge exists between two nodes if their corresponding values of $UY$ differ by Hamming distance 1. Thus, our graph contains up to $2n^2$ nodes and $n^2 \cdot 2n^2$ edges.

Specifically, given a value of $U$ obtained by the output of Algorithm 1 we attempt to solve the blind decoding problem by solving the following program

$$\begin{align*}
\text{maximize} & \quad \log |\det U| \\
\text{s.t.} & \quad UY_g = \pm 1 \\
& \quad ||UY_b||_\infty \leq 1.
\end{align*}$$

This is a non-linear mixed-integer program and we will attempt to optimize it as such. We do so by flipping the signs of individual entries of $UY_g$ in an attempt to hillclimb towards an optimal value of $U$ while allowing for backtracking when we reach a local optimum. Before we discuss our approach to solving it, we give state several important facts about the program.

Claim 1. All global optima of (4)–(5) are also global optima of (7)–(2).

**Proof.** We know that (1)–(2) contains global maxima such that $UY \in \{-1,+1\}^{n \times k}$ [1] Lemma 3. Any $U$ such that $UY \in \{-1,+1\}^{n \times k}$ will clearly be feasible in both (1)–(2) and (4)–(5). So there must be a $U$ that maximizes (1)–(2) that also maximizes (4)–(5). Further, any $U$ that is feasible in (4)–(5) is clearly also feasible in (1)–(2). So all $U$ that maximize (4)–(5) will also maximize (1)–(2).

We note that the converse of this statement is not true: when $n$ is such that no Hadamard matrices exist, (1)–(2) may contain (non-strict) optima that are not feasible in (4)–(5), see [1]. We also note that we have not ruled out the possibility that the additional constraints imposed in (4)–(5) introduce local optima that are not present in (1)–(2). Empirically, we have not observed such optima for $n < 6$. Further, for $n \geq 6$, the fraction of local optima that are encountered while optimizing (4)–(5) is consistent with the fraction of optima that are local in (1)–(2). This suggests that either such spurious optima do not exist or are insignificant in number. However, we defer on obtaining analytic results supporting this claim.

B. Tableau Formation

We use the tableau data structure, commonly used to implement the simplex algorithm, to efficiently allow us to ‘hop’ between feasible values of $U$ and flip a single entry of $UY_g$. In this subsection, we describe how to formulate this tableau in a process that closely follows [17].

Given an output of Algorithm 1, $U$, we construct a linear program (LP) that is the first-order approximation of (4)–(5). To do this, we simply replace the objective function with the gradient of the objective function, which in this case is $U^\top$. We do not fully solve this LP as the objective function must be updated after each simplex hop. In order to form a traditional simplex tableau, the problem must be expressed in standard form. An LP in standard form optimizes a linear functional over non-negative vector $x$ subject to a series of equality constraints, explicitly, for some $b, c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, we aim to find $x \in \mathbb{R}^n$ according to

$$\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0.
\end{align*}$$

In order to express the constraints in (5) in standard form, we must replace each entry $u_{i,j}$ with a pair of variables constrained such that $u_{i,j} = x_{i'} - x_{i''}$ and $x_i \in \mathbb{R}_+$ for all $i$. The $\ell_\infty$ constraints in (5) can be replaced by a series of inequality constraints as described in Section 1. Each of these inequality constraints can further be replaced by a single equality constraint with the addition of a slack variable; that is, a constraint in the form $\sum_i a_i^{(i)} x_i \leq b_i$ becomes $\sum_i a_i x_i + x_j = b_i$ for some $x_j \geq 0.$ If $Y_g$ has $l$ columns, then we are left with a set of $2nk$ linear equations with $2(n+l)$ variables.

The final step in forming a tableau is to solve for all $x$ variables to obtain a suitable BFS. First, for each $x_{i'}$ and
to perform the necessary calculations efficiently; changing an
\( x_{i,j} \) such that \( u_{i,j} = x_{i} - x_{j} \), we must set exactly one \( x \)
variable to zero such that the remaining \( x \) variable is positive
(that is, if \( u_{i,j} \) is positive, then \( x_{i} = u_{i,j} \) and \( x_{j} = 0 \)). We
then perform Gauss-Jordan elimination on the remaining linear
equations in order to solve for the remaining \( x \) variables. We note
that the sparse structure of the tableau can be exploited to
efficiently perform this elimination, see \([17]\) for a more
complete description of this process.

\section*{C. Searching the Feasible Region}

Hopping to an adjacent vertex of the feasible region involves
flipping exactly one entry of \( \bar{U} \bar{Y} \) between its two possible
extrema, namely \( \{-1, +1\} \). The simplex tableau allows us
to perform the necessary calculations efficiently; changing an
entry of \( \bar{U} \bar{Y} \) affects only a single row of \( \bar{U} \), implying that each
pivot involves manipulating only a small subset of the linear
equations and variables that make up the full tableau. Indeed,
computing the corresponding change in \( \bar{U} \) for each neighbor
can be done in only \( O(n) \) operations. We defer to existing
literature on the simplex algorithm (for example \([17]\)) for a
description of the pivoting process. Instead, in this subsection
we focus on the process of efficiently selecting an appropriate
vertex to pivot to at each step of the depth-first search.

Unlike ordinary simplex, the process of selecting a pivot is
slightly more involved in our algorithm. Each vertex has at
most \( n^2 \) feasible neighbors. The structure of the tableau allows
us to easily determine the change that hopping to each of these
vertices will induce on \( \bar{U} \). Suppose we update the \( i \)th row of
\( \bar{U} \) such that \( u_{i} = u_{i} + \Delta \) for some \( \Delta \in \mathbb{R}^{1 \times n} \), meaning
\( \bar{U} = \bar{U} + \bar{e}_{i} \Delta \). We can use the matrix determinant lemma
(\([18]\)) to compute the corresponding change in the objective
function as

\[
\log | \det (\bar{U} + \bar{e}_{i} \Delta) | = \log | (1 + \Delta \bar{U}^{-1} \bar{e}_{i}) \det (\bar{U}) |
\]

\[
= \log | (1 + \Delta \bar{U}^{-1} \bar{e}_{i}) | + \log | \det (\bar{U}) |
\]

(6)

Because the logarithm function is monotonic, we can easily
predict the change this hop will impose on the objective
function by simply considering the value of \( | 1 + \Delta \bar{U}^{-1} \bar{e}_{i} | \).
This requires only \( O(n) \) operations. The process of inspecting
all \( n^2 \) neighbors and subsequently pivoting to an optimal
choice can be accomplished in only \( O(n^3) \) operations. More-
over, because each pivot imposes a rank-one change in \( \bar{U} \),
\( \bar{U}^{-1} \) can be efficiently updated via the Sherman-Morrison
inverse formula \([19]\) after each hop, meaning \( \bar{U}^{-1} \) need not be
computed from scratch each step.

During the vertex hopping process, we select the vertex in
the direction of the maximal positive gradient that has not
already been visited, continuing in a depth-first-search manner
until a global optimum has been located or a preset hop limit
has been exhausted. For many values of \( n \), there may exist
local optima; for this reason, we allow the solver to hop to
neighboring vertices where the objective function is equal or
less in value. For most values of \( n \), we can detect when
we have reached a global optimum, allowing the solver to
terminate without enumerating all non-singular vertices of the
feasible region. Our early termination criteria are discussed in
Section \([VT-D]\).

In order to efficiently backtrack when we reach a local
optimum, we keep a complete snapshot of the tableau at each
of the previously visited vertices. While we do allow the solver
to visit previously unvisited vertices that decrease the objective
function, we do not allow the solver to traverse vertices where
\( \det \bar{U} = 0 \). This is because the gradient of the objective
function is undefined at these vertices. While this may restrict
the search space of the solver, this is not frequently an issue.
We discuss cases where this becomes an issue in Section \([VT-E]\).

Finally, we note that considering other criteria to select the
next vertex in our search is a topic of future research. For
example, it may be possible to further optimize the vertex
hopping process by simply hopping to the first neighbor
encountered that increases the objective function rather than
exhaustively checking all \( n^2 \) neighbors. Additionally, when
the fraction of ‘bad’ columns is large, one may consider positively
weighting vertices which cause more columns to be ‘good’.
For \( n \leq 5 \), our current vertex hopping strategy rarely requires
more than one or two hops to reach a global optima. However,
as \( n \) grows, for certain values of \( k \) the majority of the time to
obtain the solution is consumed by the vertex hopping process.
Thus, improved vertex selection criteria may be necessary to
further reduce the runtime of our algorithm for \( n > 5 \).

\begin{algorithm}
\caption{Simplex with Backtracking}
\textbf{Input:} \( \bar{U} \) such that \( \bar{U} \bar{Y} \) is full rank.
\textbf{Output:} \( \hat{X} \) or error.
\begin{algorithmic}[1]
\State form simplex tableau
\Repeat
\State compute objective function at all neighboring vertices
\If {optimum reached}
\State if global optimum then
\State return \( \hat{X} = \bar{U} \bar{Y} \)
\Else
\State backtrack to first vertex with unvisited neighbor
\EndIf
\EndIf
\State pivot to the largest feasible unvisited neighbor
\State update \( \bar{U}^{-1} \).
\Until {state space exhausted}
\State return error
\end{algorithmic}
\end{algorithm}

\subsection*{D. Stopping Criteria}

Since we do not know \( \bar{A} \), nor the value of \( | \det \bar{A} | \), the value
of \( \det \bar{U} \) alone does not tell us whether \([1\hspace{.1cm}2]\) has obtained
a global optimum. However, the determinant of any square
matrix with values \( \pm 1 \) can only take on a discrete set of values
(see \([16]\) or \([20]\)). As a result, the only values that \( | \det \bar{U} | \) can
take is simply this spectrum scaled by some unknown constant,
namely \( \det \bar{A}^{-1} \). In other words, \([1\hspace{.1cm}2]\) is maximized when
the following quantity obtains the maximum determinant for
any ±1-valued matrix of dimension \( n \):

\[
\max_{\tilde{Y} \in \mathbb{R}^{k \times n}} |\det U \tilde{Y}| \tag{7}
\]

\[
s.t. \ cols(\tilde{Y}) \subseteq cols(Y). \tag{8}
\]

Solving this problem seems like a difficult combinatorial optimization task so we do not rely on solving it directly. However, when we are at an optimum, we can often determine the value of (7) by inspecting the value of the objective function at neighboring vertices. Indeed, for certain values of \( n \), we may uniquely determine when we have reached a global optimum by considering the relative change between the value of the objective function at the optimum and at its \( n^2 \) neighbors.

In this subsection, we provide necessary and sufficient conditions to determine when Algorithm 3 has reached a global optima, for \( n \in \{2, \ldots, 6, 8\} \). In Appendix \( \mathcal{A} \) we prove that these conditions are sufficient up to \( n = 6 \); the condition given for \( n = 8 \) is conjectured to be sufficient based on empirical results.

As stated in Section VI-A, each vertex may have up to \( n^2 \) neighbors, although not all these neighbors may be feasible. Regardless of whether or not these vertices are feasible, we may still compute the objective function at these neighboring values using the process outlined in Section VI-C. The following criteria, describing the value of the objective function at all \( n^2 \) neighboring vertices, may be applied to uniquely determine when Algorithm 3 has reached a global optimum.

- For \( n \leq 5 \), there are no local optima. In these cases, if all \( n^2 \) neighbors of \( UY \) decrease the value of the objective function, then \( UY \) must be a global optimum.

- For \( n = 6 \), at any global optimum, the value of the objective function at all 36 neighbors will take on exactly 3 distinct values. Further, only global optima have neighbors such that the value of the objective function decreases to a fraction of 4/5 of the optimal value.

- For \( n = 8 \), if all 64 neighbors of \( UY \) decrease the value of the objective function to a fraction of 3/4 of the optimal value, then \( UY \) must be a global optimum.

- For \( n = 10 \), 20 neighbors of \( UY \) must decrease the value of the objective function by a fraction of 1/3 the optimal value while the remaining 80 neighbors must decrease the value by 1/6.

- For \( n = 12 \), all 144 neighbors of \( UY \) must decrease the objective function by 1/6 of the optimal value.

### E. Causes of Failure

In this subsection, we consider when Algorithms 1 (Vertex Finding) and 3 (Simplex with Backtracking) fail to properly terminate. Provided that these algorithms are given proper inputs, the probability that they succeed, for typical values of \( n \) and \( k \), is high. In cases where we do encounter a failure, we find that rerunning the algorithm with a new starting point (\( U_0 \)) is often sufficient to recover from this failure. For both Algorithms 1 and 3 we provide the average number of calls required to obtain a solution to the blind decoding problem for various values of \( n \) and \( k \) in Table II along with the average runtime of each algorithm. We outline the reasons both Algorithms 1 and 3 fail in the remainder of this subsection.

Although not depicted in Table II, we note that the number of calls to each subroutine is highly dependent on the value of \( k \). This is evidenced in Figure 4 which plots the average runtime of our solver, normalized by \( k \), for \( n = 6, 8, 10, \) and 12. As previously discussed, an optimistic estimate of the runtime of our algorithm is \( O(n^4k) \). If our algorithm performed near this estimated runtime, we would expect these lines would be constant, i.e. that the runtime would scale linearly with \( k \). For \( n \geq 6 \) and small values of \( k \), our algorithm does not appear to perform near the best-case complexity. This is because, for large \( n \) and small \( k \), the output of Algorithm 1 is often insufficient to produce a \( Y_g \) that is full rank and thus we require many calls to this subroutine to produce a suitable \( Y_g \). We further note that in Figure 4 at large \( k \), the runtime appears to grow approximately quadratically by \( k \) rather than linearly as predicted by our best-case analysis. This is because our current implementation does not fully exploit the sparsity of the tableau matrix; doing so would in fact impose a performance penalty for small \( k \). We now discuss the cases in which our subroutines fail in greater detail.

1) Insufficient ‘Good’ Columns: In order to form a simplex tableau, we must find at least \( n \) linearly independent columns that are contained in \( \{-1, +1\}^n \). When \( n = k \) and \( X \) is

### TABLE II

| \( n \) | \( k \) | Avg. Calls | Time per Call (s) | Avg. Calls | Time per Call (s) |
|---|---|---|---|---|---|
| 2 | 8 | 1.01 | 5.33e-6 | 1.00 | 1.29e-5 |
| 3 | 13 | 1.01 | 1.89e-5 | 1.00 | 4.57e-5 |
| 4 | 18 | 1.03 | 5.68e-5 | 1.00 | 1.15e-4 |
| 5 | 18 | 1.28 | 9.78e-5 | 1.06 | 1.63e-4 |
| 6 | 22 | 1.75 | 2.31e-4 | 1.20 | 3.69e-4 |
| 8 | 30 | 4.47 | 6.11e-4 | 1.41 | 1.18e-3 |

![Fig. 4. The average time to solution of our algorithm normalized by \( k \), the number of samples. For small values of \( k \), the runtime is dominated by failures of Algorithm 1. For large \( k \), the runtime is dominated by the vertex hopping process.](image-url)
maximal, we are guaranteed to find such a result. However, for \( k > n \), this is no longer guaranteed. We find that for small \( n \), as well as for all \( n \) when \( k \) is large, we obtain such a result with high probability. However, we find that for large \( n \) and small \( k \), the probability that Algorithm 1 fails to produce a suitable output becomes non-negligible.

Our empirical results suggest that for \( n \leq 5 \), Algorithm 1 is certainly sufficient to initialize Algorithm 3. It is an active area of research to determine whether or not alternative initialization methods may help improve the average runtime of our approach for larger \( n \).

2) Trap Cases: For \( n = 6 \) and \( n = 8 \), it is possible to obtain a BFS that lies in a component of the graph of vertices that does not contain a global optimum. We refer to such cases as “trap cases.” In these cases the solver will enumerate the entire component and then exit without returning a solution to the blind decoding problem. For both \( n = 6 \) and \( n = 8 \), empirically, we see that the odds of finding such trap cases decreases as \( k \) increases. Roughly, this occurs because either portions of these components, or the entire component, becomes infeasible.

We find that when the solver is presented a BFS that lies on a graph component with a global optimum, for \( n \leq 8 \), the solver typically only requires a very small number of hops (rarely more than 4) in order to find this solution. However, we note that as \( n \) increases, the number of vertices of the feasible region grows exponentially fast; and we find that, for \( n = 8 \), a trapped subspace may contain tens of thousands of non-singular vertices. The default behavior of the algorithm is to enumerate this entire subspace in attempt to find a global optimum. As a simple heuristic to detect a trap cases, we simply limit the maximum number of vertices that the solver may visit in a single attempt. If the solver exceeds this limit, we assume that the solver has encountered a trap case, and we proceed by returning to Algorithm 1 and finding a new BFS. Empirically, we find that setting this limit to \( 2nk \) is sufficiently large to almost always avoid falsely detecting a trap case. It is an open question to determine if there exist more intelligent methods to recover from a trap case rather than restarting the solver from scratch.

3) False Trap Cases: For all values of \( n \), we find that there is a small probability that Algorithm 3 will incorrectly terminate after enumerating the entire state space, thus presenting what appears to be a trap case. These ‘false traps’ are caused by numerical instability; the solver may deem a neighboring vertex to be infeasible because the values of one or more entries of \( \text{UY} \) exceeds ±1 by more than a predefined floating point threshold.

For \( n < 5 \), the odds of encountering a false trap is less that one per one thousand with this probability increasing slightly as \( n \) grows further. When such an error occurs, we typically find that the channel gain matrix is poorly conditioned \((\kappa > 10^5)\). Surprisingly, even with such poorly conditioned channels, we can often recover from this type of error by simply obtaining a new BFS and trying again. This behavior indeed seems difficult to avoid for extremely ill-conditioned channels. Because this behavior occurs so infrequently and only in poorly conditioned channels, we believe that simply restarting is a sufficient solution and we do not need to consider further optimizations.

VII. ROBUST DECODING

A. Algorithm

So far, we have only considered solving the blind decoding problem in the limit of high SNR. We now turn our attention to how to robustly solve the blind decoding problem in the presence of AWGN. We make no claims that the technique provided here is optimal in terms of BER performance; it almost certainly is not. However, our technique works well empirically and is computationally efficient. Our treatment of this algorithm is entirely empirical; an analytic treatment of the performance of our algorithm in the presence of noise is a topic of ongoing research.

In order to understand how to adapt to noise, we consider the output of Algorithm 1 in the noiseless case in more detail. Empirically, for any \((n, k)\), when Algorithm 1 terminates, the vast majority of entries of \( \text{UY} \) are contained in the set \( \{-1, +1\} \). A small proportion, on the order of 1%, of entries are 0-valued and an even smaller proportion (roughly two out of 10000) appear uniformly distributed within \(( -1, +1 \) \( \setminus \{0\} \).

Empirical results describing the distribution of the entries of \( \text{UY} \) for \( n = 4 \) and various values of \( k \) is shown in Figure 5. When AWGN is present, the behavior of Algorithm 1 changes drastically because the columns of \( \text{Y} \) no longer have such strong linear dependence. With near certainty, the output of Algorithm 1 will have only \( n^2 \) entries that are exactly \( \{-1, +1\} \), and, as \( k \) grows, this means that \( Y_g \) will almost never be full rank and we will be unable to proceed with Algorithm 3.

To address this issue, we add an additional rounding step to Algorithm 1. If entries of \( \text{UY} \) are within some tolerance,
of \{-1,0,1\}, we simply adjust \( Y \) to effectively round off the corresponding entry of \(UY\). Explicitly, we compute each a matrix \(\Delta\) where the \(ij\)th entry is given by

\[
\Delta_{ij} = \begin{cases} 
(UY)_{ij} + 1 & |(UY)_{ij} + 1| < \delta \\
(UY)_{ij} - 1 & |(UY)_{ij} - 1| < \delta \\
(UY)_{ij} & |(UY)_{ij}| < \delta \\
0 & \text{otherwise}.
\end{cases}
\]  

(9)

\( Y \) is then updated by computing \( Y = Y - U^{-1}\Delta \). Performing this rounding step only once is typically not sufficient to find a value of \( Y_g \) that is sufficient to construct a tableau. Indeed, rounding only once is often insufficient when \( X \) has columns that are identical up to a sign. In these cases, \(Y_g\) will be deficient in rank by the number of identical columns of \( X \). More independent columns of \( Y_g \) may be found at this point by returning to the main procedure of Algorithm \(\text{IV} \) after rounding.

The full rounding process is presented in Algorithm \(\text{IV} \). In practice, we find that the \(\text{for}\) loop contained in this algorithm will often terminate after only one or two iterations. Repeating the rounding process beyond \( n \) times will have no effect; after completing \( n \) iterations of the process we are guaranteed that the set of active constraints will be full rank.

**Algorithm 4** Vertex Finding in the presence of AWGN

**Input:** \( Y \)

**Output:** \( Y, U \) which is a BFS.

1. Draw \( U_0 \) at random as usual
2. \( Y_0 = Y \)
3. for \( i = 0, i < n, i += 1 \) do
4. \( U_{i+1} = \text{Algorithm IV}(U_i, Y_i) \)
5. if \( \|U_{i+1} - U_i\|_\infty < \delta \) then
6. break
7. end if
8. Compute \( \Delta \) as in eq. (9).
9. \( Y_{i+1} = Y_i - U_{i+1}\Delta \)
10. end for
11. return \( U_i, Y_i \).

**B. Choosing \(\delta\) and BER Performance**

We now turn our attention to discussing the BER performance of the vertex hopping algorithm using Algorithm \(\text{IV} \). We first consider the behavior of this algorithm in the high SNR limit. From Figure \(\text{V} \) we see that roughly 2 out of 10,000 of the entries are distributed in the interval \((-1,1)\). When these entries are within \( \delta \) of \( \pm 1 \) then Algorithm \(\text{IV} \) will incorrectly force these entries to \( \pm 1 \). This may result in recovering an estimate of \( A^{-1} \) that is not equivalent up to an ATM. This behavior explains the noise enhancement observed at high SNR in Figure \(\text{IV} \).

In the presence of AWGN, we observed that the BER is effectively constant for a given value of \( \delta \). Instead, with a fixed value of \( \delta \), as SNR decreases, the odds that the solver will complete decays. This is because, if \( \delta \) is small compared to the noise variance, the rounding procedure described above is unlikely to have any effect. As a result, it is unlikely that Algorithm \(\text{IV} \) will yield a value of \( Y \) with a full rank \( Y_g \). The odds that Algorithm \(\text{IV} \) produces a suitable output as a function of SNR for various values of \( \delta \) is presented in Figure \(\text{VI} \).

Unlike the trap cases discussed in Section \(\text{VI-E} \), the odds of Algorithm \(\text{IV} \) succeeding is largely dependent on the input \( Y \) and not the initial choice of \( U_0 \). In other words, if the solver fails at low SNR, it is unlikely to succeed if it is run again with a different choice of \( U_0 \). In practice, in the low SNR regime, we run several iterations with different choices of \( U_0 \) to account for failures due to trap cases, but if these attempts fail we must either declare an erasure or raise the value of \( \delta \). This behavior allows us to choose the value of \( \delta \) based requisite bit- and block-error rates. Figure \(\text{VII} \) shows the average BER as a function of \( \delta \). These results are averaged over all SNR values ranging from 10dB to 30dB.

Finally, we note that, in addition to the noise variance, the condition number of the channel gain matrix also plays a large role in both the success probability depicted in Figure \(\text{VI} \) and the BER performance. It is certainly possible to pick \( \delta \) for example, based in part on the singular values of \( Y \), rather than exclusively on the SNR. We did not explore how to exploit this relationship as computing the SVD of \( Y \) adds significant complexity at the receiver. Further, if the distribution of the channel gain matrix is such that the variance of its condition number is small over each block then such an approach would be of little benefit.

**VIII. Conclusion**

In this work, we have presented a ‘vertex hopping’ algorithm that can efficiently and reliably perform blind decoding of BPSK MIMO symbols in the presence of AWGN. We present the problem of decoding as a non-linear mixed-integer
program and leverage techniques from solving linear programming to solve it. Our vertex hopping algorithm consists of a two-step process; we must first find a suitable vertex of the feasible region that describes the integer program. Once this vertex is found we hop between vertices of the feasible region in an attempt to search for a global optimum.

Empirically, we show that this technique is both efficient and reliable for MIMO systems as large as \( n = 8 \). Our technique still works beyond \( n = 8 \), but the underlying non-convex optimization problem appears to become computationally difficult as \( n \) grows. For small \( n \), the vertex hopping algorithm is both efficient and practical in terms of sample and computational complexity. At low SNR, our technique performs comparably to zero-forcing decoding and at all SNRs outperforms ML decoding with as little as 1% CSI error.

The algorithm presented in this work motivates a suite of future research topics. Many of these topics are related to understanding and improving the performance of our algorithm in a variety of operating conditions. Examples include: developing optimizations to exploit the structure present in complex channel gain matrices, studying the performance of our algorithm under higher modulation orders, and considering rectangular channels. Further, more research is warranted at the wireless-systems level to understand how to best use our algorithm to build high-rate, reliable MIMO systems that operate in environments with rapidly changing CSI.

**Appendix A**

**Stopping Criteria**

In this appendix, we prove that the criterion given in Section [VI-D] are indeed necessary and sufficient to determine whether a value of \( U \) is at a global optimum. For simplicity of exposition, we simply consider the case \( k = n \); the arguments contained in this section indeed still hold when \( k > n \) as we can still compute the objective function of all \( n^2 \) neighboring vertices regardless of whether or not they are feasible.

**Theorem 1.** For \( n \leq 6 \), a global optimum can be detected based on the value of the objective function obtained on each of its \( n^2 \) neighbors.

This theorem follows trivially for \( n \leq 5 \) due to the fact that the only optima contained on the vertices of the feasible region are in fact global in these cases. This is proven in [1] and [13]. Before proving this theorem for \( n = 6 \), we formalize the discussion in Section [VI-D] regarding the optima of (7)–(8). Define the set \( D = \{ \{ \det \mathbf{X} : \mathbf{X} \in \{-1,+1\}^{n \times n} \} \} \), which in [16] is referred to as the spectrum of possible determinants. Since, in the noiseless case, \( \det U Y = \det U A X \), and \( |\det \mathbf{X}| \in D \), this implies the following claim:

**Claim 2.** The value of \( |\det U| = D |\det A^{-1}| \) for some \( D \in D \).

We say that a \( \pm 1 \)-valued matrix is maximal if it obtains the maximum determinant amongst all matrices constrained to \( \{-1,+1\}^{n \times n} \). Suppose there are \( N \) distinct maximal matrices, then we denote these matrices as \( \mathbf{X}_1, \ldots, \mathbf{X}_N \).

**Definition 1.** Equivalence of matrices. Two matrices \( \mathbf{X}_1, \mathbf{X}_2 \) are weakly equivalent if, for some \( \mathcal{T}_1, \mathcal{T}_2 \in \mathcal{T} \), \( \mathbf{X}_1 = \mathcal{T}_1 \mathbf{X}_2 \mathcal{T}_2 \). If \( \mathbf{X}_1 = \mathcal{T}_1 \mathbf{X}_2 \mathcal{T}_2 \), then \( \mathbf{X}_1, \mathbf{X}_2 \) are strongly equivalent.

Strong equivalence is the same as being equivalent up to an ATM, which only allows permutation and negation of columns. All solutions to the blind decoding problem are strongly equivalent. However, matrices that are weakly equivalent to \( \mathbf{X} \) may have both permuted rows and columns. Such matrices are not necessarily solutions to the blind decoding problem and may correspond to spurious optima.

We now consider the poset formed by taking a single vertex and its \( n^2 \) neighbors and imposing an ordering on the graph. Specifically, the poset \( \{ \mathcal{X}, \preceq \} \) formed by taking the set, \( \mathcal{X} \), containing \( U Y \) and its \( n^2 \) neighbors, and imposing the ordering where \( \mathbf{A} \leq \mathbf{B} \) implies \( |\det \mathbf{A}| \leq |\det \mathbf{B}| \).

**Lemma 1.** Suppose \( n \) is such that all maximal \( \pm 1 \)-valued matrices are weakly equivalent and that there are \( N \) maximal matrices. Then all posets \( \{\mathcal{X}_1, \preceq\}, \ldots, \{\mathcal{X}_N, \preceq\} \) are isomorphic.

**Proof.** Consider the mapping \( \phi_{k,l} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n} \) given by \( \mathbf{X}_i \mapsto \mathcal{T}_k \mathbf{X}_i \mathcal{T}_l \), for some \( \mathcal{T}_k, \mathcal{T}_l \in \mathcal{T} \). This mapping is one-to-one since all elements of \( \mathcal{T} \) are square and full rank. This mapping also preserves the ordering of the poset \( \{\mathcal{X}_i, \preceq\} \). This is because permuting rows or columns of a matrix can only change the sign of its determinant, and likewise for negating its rows or columns; hence, for any \( \mathbf{X}, \mathcal{T}_k, \mathcal{T}_l \), we have \( |\det \mathbf{X}| = |\det \mathcal{T}_k \mathbf{X} \mathcal{T}_l| \). If \( \mathbf{X}_j = \phi_{k,l}(\mathbf{X}_i) \), then this implies that the poset \( \{\mathcal{X}_i, \preceq\} \) is isomorphic to \( \{\mathcal{X}_j, \preceq\} \). Since \( n \) is such that there is only one weak equivalence class of maximal determinant matrices, this implies that for any \( \mathbf{X}_i \) and \( \mathbf{X}_j \), there exists a \( k, l \) such that \( \mathbf{X}_j = \phi_{k,l}(\mathbf{X}_i) \). Thus, the posets \( \{\mathcal{X}_i, \preceq\}, \{\mathcal{X}_j, \preceq\} \) are isomorphic for all \( i, j \).\end{proof}
Up to \( n = 10 \), and for several values beyond \( n > 10 \), all maximal matrices are weakly equivalent (see [21]): Lemma [1] holds for these cases. Beginning at \( n = 6 \), local optima exist in [1]–[2]. However, the following lemma shows that the poset obtained at any global optima in this case is uniquely identifiable.

Lemma 2. For \( n = 6 \), the poset \((\mathcal{X}, \preceq)\) obtained at a global optimum is uniquely identifiable.

Proof. For \( n = 6 \), we have \( D = \{0, 32, 64, 128, 160\} \). By Lemma [1] and the fact that all maximal matrices at \( n = 6 \) are weakly equivalent, we can consider the neighbor pattern of any one maximal vertex. We now require the following claim, which may be verified by inspecting all 36 neighbors of any maximal \( n = 6 \) matrix.

Conjecture 2. For \( n = 10 \), a vertex is maximal if and only if 20 of its 100 neighbors decrease the objective function by a factor of \( 1/3 \) and 80 of its 100 neighbors decrease the objective function by a factor of \( 1/6 \).

Conjecture 3. For \( n = 12 \), a vertex is maximal if and only if all 144 of its neighbors decrease the value of the objective function by a factor of \( 1/6 \).

References

[1] T. Dean, M. Wootters, and A. Goldsmith, “Blind joint MIMO estimation and decoding,” IEEE Trans. Inf. Theory, in press. Preliminary version at https://arxiv.org/abs/1802.01049.

[2] J. G. Andrews, S. Buzzi, W. Choi, et al., “What will 5G be?,” IEEE J. Sel. Areas Commun., vol. 32, no. 6, pp. 1065–1082, 2014.

[3] F. Boccardi, R. W. Heath, A. Lozano, et al., “Five disruptive technology directions for 5G,” IEEE Commun. Mag., vol. 52, no. 2, pp. 74–80, 2014.

[4] T. S. Rappaport, S. Sun, R. Mayzus, et al., “Millimeter wave mobile communications for 5G cellular: It will work!,” IEEE Access, vol. 1, pp. 335–349, 2013.

[5] A. Osseiran, F. Boccardi, V. Braun, et al., “Scenarios for 5G mobile and wireless communications: the vision of the METIS project,” IEEE Commun. Mag., vol. 52, no. 5, pp. 26–35, 2014.

[6] E. Telatar, “Capacity of multi-antenna gaussian channels,” Trans. on Emerging Telecommunications Technologies, vol. 10, no. 6, pp. 585–595, 1999.

[7] A. Goldsmith, S. A. Jafar, N. Jindal, and S. Vishwanath, “Capacity limits of MIMO channels,” IEEE J. Sel. Areas Commun., vol. 21, no. 5, pp. 684–702, 2003.

[8] L. Zheng and D. N. C. Tse, “Communication on the grassmann manifold: A geometric approach to the noncoherent multiple-antenna channel,” IEEE Trans. Inf. Theory, vol. 48, no. 2, pp. 359–383, 2002.

[9] S. Talwar, M. Viberg, and A. Paulraj, “Blind separation of synchronous co-channel digital signals using an antenna array. I. algorithms,” IEEE Trans. Signal Process., vol. 44, no. 5, pp. 1188–1197, 1996.

[10] L. K. Hansen and G. Xu, “A hyperplane-based algorithm for the digital co-channel communications problem,” IEEE Trans. Inf. Theory, vol. 43, no. 5, pp. 1536–1548, 1997.

[11] 3GPP, “TS 36.213 evolved universal terrestrial access (E-UTRA); physical layer procedures,” ETSI Tech. Rep., 2010.

[12] IEEE Computer. Society, “Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications,” Std 802.11ac-2013, 2013.

[13] T. Dean, J. Perlstein, M. Wootters, and A. Goldsmith, “Blind decoding performance guarantees in the \( n = 5 \) case,” in preparation.

[14] S. Boyd and L. Vandenberghe, Convex optimization. Cambridge university press, 2004.

[15] R. M. Karp, “Reducibility among combinatorial problems,” in Complexity of computer computations, pp. 85–103, Springer, 1972.

[16] W. P. Orrick and B. Solomon, “Spectrum of the determinant function,” http://www.indiana.edu/~maxdet/spectrum.html.

[17] C. H. Papadimitriou and K. Steiglitz, Combinatorial optimization: algorithms and complexity. Courier Corporation, 1998.

[18] J. Ding and A. Zhou, “Eigenvalues of rank-one updated matrices with some applications,” Applied Mathematics Letters, vol. 20, no. 12, pp. 1223–1226, 2007.

[19] J. Sherman and W. J. Morrison, “Adjustment of an inverse matrix corresponding to a change in one element of a given matrix,” The Annals of Mathematical Statistics, vol. 21, no. 1, pp. 124–127, 1950.

[20] F. Macwilliams and N. Sloane, The Theory of Error-correcting Codes. North-Holland, 2006.

[21] W. D. Smith, “Studies in computational geometry motivated by mesh generation,” tech. rep., Princeton Univ., NJ (USA), 1989.