Matrix Rigidity: Matrix Theory from the Viewpoint of Parameterized Complexity

Fedor V. Fomin[1], Daniel Lokshtanov[1], S. M. Meesum[2], Saket Saurabh[1,2], and Meirav Zehavi[1]

1 **University of Bergen, Norway.** `{fomin,daniello,meirav.zehavi}@ii.uib.no`
2 **The Institute of Mathematical Sciences, Chennai, India.** `{meesum,saket}@imsc.res.in`

Abstract

The *rigidity* of a matrix A for a target rank r over a field $\mathbb{F}$ is the minimum Hamming distance between A and a matrix of rank at most r. Rigidity is a classical concept in Computational Complexity Theory: constructions of rigid matrices are known to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. In this paper, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity. Naturally, given a parameter k, the Matrix Rigidity problem asks whether the rigidity of A is at most k. In the case when $\mathbb{F} = \mathbb{Q}$ or if the replaced entries must be integers, it is not even known whether the problem is decidable. Surprisingly, we show that in case $\mathbb{F} = \mathbb{R}$ or $\mathbb{F}$ is any finite field, this fundamental problem is fixed-parameter tractable with respect to $k + r$. To this end, we present a simple yet powerful dimension reduction procedure, which we believe to be a valuable primitive in future studies of problems of this nature. We further employ central tools in Real Algebraic Geometry, which are not well known in Parameterized Complexity, thus establishing their ability to form a bridge between Matrix Theory and Parameterized Complexity. In particular, we view the output of our dimension reduction procedure as an algebraic variety. Our main results are complemented by a W[1]-hardness result and a subexponential-time parameterized algorithm for a special case of Matrix Rigidity, highlighting the different flavors of this problem.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases Matrix Rigidity, Parameterized Complexity, Linear Algebra

1 Introduction

The *rigidity of a matrix* is a classical concept in Computational Complexity Theory, which was introduced by Grigoriev [8, 9] in 1976 and by Valiant [24] in 1977. Constructions of rigid matrices are known, for instance, to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. The few papers that do consider such properties are restricted to the very special case of adjacency matrices, and therefore they are primarily studies in Graph Theory rather than Matrix Theory [17, 18]. In this paper, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity.

Formally, given a matrix A over a field $\mathbb{F}$, the rigidity of A, denoted by $\mathcal{R}_A^{\mathbb{F}}(r)$, is defined as the minimum Hamming distance between A and a matrix of rank at most r. In other words, $\mathcal{R}_A^{\mathbb{F}}(r)$ is the minimum number of entries in A that should to be edited in order to obtain a matrix of rank at most r. Naturally, given a parameter k, the MATRIX RIGIDITY problem asks whether $\mathcal{R}_A^{\mathbb{F}}(r) \leq k$. The case when $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not even known whether the problem is decidable [21]. We therefore focus on the cases where $\mathbb{F} = \mathbb{R}$ or $\mathbb{F}$ is any finite field. Formally, we study the following forms of MATRIX RIGIDITY. Here, FF MATRIX RIGIDITY is not restricted to a specific finite field $\mathbb{F}_p$, but includes $\mathbb{F}_p$ as part of the input.

REAL MATRIX RIGIDITY **Parameter:** r, k
Input: A matrix A with each entry an integer, and two non-negative integers r, k.
Question: Is $\mathcal{R}_A^{\mathbb{R}}(r) \leq k$?

FF MATRIX RIGIDITY **Parameter:** p, r, k
Input: A finite field $\mathbb{F}_p$ of order p, a matrix A over $\mathbb{F}_p$, and two non-negative integers r, k.
Question: Is $\mathcal{R}_A^{\mathbb{F}_p}(r) \leq k$?

Valiant [24] presented the notion of the rigidity of a matrix as a means to prove lower bounds for linear algebraic circuits. He showed that the existence of an $n \times n$ matrix A with $\mathcal{R}_A^{\mathbb{F}}(\epsilon n) \geq n^{1+\delta}$ would imply that the linear transformation defined by A cannot be computed by any arithmetic circuit having size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$ in which each gate of the circuit computes a linear combination of its inputs. Later, Razborov [20] (see [13]) established relations between lower bounds on rigidity of matrices over the reals or finite fields and strong separation results in Communication Complexity. Although many efforts have been made in this direction [7, 22, 14, 11] (this is not an exhaustive list), proofs of separation lower bounds for explicit families of matrices still remains elusive. For a recent survey on this topic we refer the reader to [15]. The formulation of the MATRIX RIGIDITY as stated in this paper was first considered by Mahajan and Sharma [16], and it was shown to NP-hard for any field by Deshpande [5].

In this paper, we study the concept of the rigidity of a matrix from a different perspective, given by the framework of Parameterized Complexity. Here, each problem instance is associated with a parameter k, and we say that a problem is *fixed parameter tractable* (FPT) if any instance (I, k) of the problem can be solved in time $\tau(k)|I|^{\mathcal{O}(1)}$, where τ is an arbitrary function of k. Throughout this paper, we use the standard notation $\mathcal{O}^*$ to hide factors polynomial in $|I|$.[1] On the one hand, to prove that a problem is FPT, it is possible to give an explicit algorithm of the required form, called a *parameterized algorithm*, which solves it. On the other hand, to show that a problem is unlikely to be FPT, it is possible to use polynomial-time reductions analogous to those employed in Classical Complexity. Here, the concept of W[1]-hardness replaces the one of NP-hardness, and we need not only construct an equivalent instance in FPT time, but also ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original instance.

A central notion in Parameterized Complexity is the one of *kernelization.* Formally, a parameterized problem Π is said to admit a *polynomial kernel* if there is a polynomial-time algorithm, called a *kernelization algorithm*, that given any instance of Π, outputs an equivalent instance of Π whose size is bounded by $\tau(k)$, where τ is a function polynomial in k and independent of $|I|$. We say that the reduced instance is a $p(k)$*-kernel* for Π. Roughly

[1] That is, $\mathcal{O}^*(\tau(k)) = \tau(k) \cdot |I|^{\mathcal{O}(1)}$.

speaking, a kernelization algorithm can be viewed as an efficient preprocessing procedure that satisfies a well defined restriction with respect to the size of its output. For more information about Parameterized Complexity in general and Kernelization in particular, we refer the reader to monographs such as [6, 3].

Finally, we remark that Meesum et al. [17] and Meesum and Saurabh [18] studied the following problems, which are related to Matrix Rigidity but are significantly simpler than Matrix Rigidity as they are restricted to graphs. Given a graph $G = (V, E)$ and two non-negative integers r, k, r-Rank Vertex Deletion (r-Rank Edge Deletion) asks whether one can delete at most k vertices (resp. edges) from G so that the rank of its adjacency matrix would be at most r, while r-Rank Edge Editing asks whether one can edit k edges in G so that the rank of its adjacency matrix would be at most r.[2] For undirected graphs, Meesum et al. [17] proved that these problems are NP-hard even if r is fixed, but can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(k \log r)})$. They also showed that r-Rank Edge Deletion and r-Rank Edge Editing can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(f(r)\sqrt{k}\log k)})$. Meesum and Saurabh [18] obtained similar results for directed graphs.

Our Contribution. In this paper, we establish that both Real Matrix Rigidity and FF Matrix Rigidity are FPT with respect to $r+k$. Specifically, we obtain the following results.

▶ **Theorem 1.1.** Real Matrix Rigidity *can be solved in time* $\mathcal{O}^*(2^{\mathcal{O}((r+k)\cdot\log(r\cdot k))})$.

▶ **Theorem 1.2.** FF Matrix Rigidity *can be solved in time* $\mathcal{O}^*(f(r,k))$ *for a function* f *that depends only on* r *and* k.

Observe that the dependency of the running times on the dimension of the input matrix is polynomial, and in the case of FF Matrix Rigidity, the dependency of the running time on p is also polynomial. Interestingly, in the case of Real Matrix Rigidity, the dependency of the running time on the maximum length (in binary) of any entry in *both* input and output matrices is polynomial! We find these results quite surprising, particularly due to the fact that in case $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not even known whether Matrix Rigidity is decidable [21]. We also show that,

▶ **Theorem 1 ($\star$[3]).** *The* FF Matrix Rigidity *problem is solvable in time* $\mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$ *for some function* f *that depends only on* r *and* p.

Here, the dependency of the running time on k is subexponential, but the dependency of the running time on p is unsatisfactory in case p is not fixed. This algorithm is based on a technically involved adaptation of ideas already used in the papers [17, 18].

To obtain our main results, we first present a dimension reduction procedure, which we believe to be a valuable primitive in future studies of problems of this nature. Our procedure is simple to describe and given an instance of Matrix Rigidity, it outputs (in polynomial time) an equivalent instance where the matrix contains at most $\mathcal{O}((r\cdot k)^2)$ entries. Furthermore, the set of entries of the output matrix is a subset of the set of entries of the input matrix. We believe, this procedure to be of interest independent of our main results as it establishes that FF Matrix Rigidity admits a polynomial kernel with respect to $r+k+p$. The simplicity of our procedure also stems from its modularity—it handles rows and columns in separate phases. On a high-level, this procedure is defined as follows. For $k+1$ steps, it repeatedly selects a set of maximum size consisting of rows that are linearly

[2] Editing an edge $\{u, v\}$ means that if $\{u, v\} \in E$ then $\{u, v\}$ is deleted, and otherwise it is added.
[3] Due to space constraints the proof will appear in the full version of the paper.

independent, where if the size of this set exceeds $r + 1$, it is replaced by a subset of size exactly $r + 1$. Each such set of rows is removed from the input matrix, and then it is inserted into the output matrix. At the end of this greedy process, rows that remain in the input matrix are simply discarded. The correctness of our procedure relies on two key insights: **(i)** if the input instance contains more than $k + 1$ pairwise-disjoint sets of rows that are linearly independent, and each of these sets is of size at least $r + 1$, then the input instance is a NO-instance; **(ii)** by the pigeonhole principle, any row discarded from the input matrix belongs to the span of at least one set of rows that cannot be edited! Having an intermediate matrix with a small number of rows, the procedure applies the exact same process to the input that is the transpose of this intermediate matrix, thus overall obtaining a matrix with a small number of entries.

Armed with our dimension reduction procedure, we tackle Real Matrix Rigidity and FF Matrix Rigidity by employing central tools in Algebraic Geometry, which are not well known in Parameterized Complexity, thus establishing their ability to form a bridge between Matrix Theory and Parameterized Complexity. For this purpose, we first recall that the rank of a matrix is at most r if and only if the determinant of all of its $(r + 1) \times (r + 1)$ submatrices is 0. Since at this point we can assume that we have a matrix containing only $\mathcal{O}((r \cdot k)^2)$ entries at hand, we may "guess" *which* entries should be edited. Yet, it is not clear *how* these entries should be edited. However, with the above observation in mind, we are able to proceed by viewing our current problem in terms of an algebraic variety. In particular, this viewpoint gives rise to the applicability of firmly established tools that determine the feasibility of a system of polynomials [1, 10].

Our main results are also complemented by a W[1]-hardness result as well as a subexponential-time parameterized algorithm for a special case of Matrix Rigidity, which overall present the different flavors of this problem and the techniques relevant to its study. We show that both Real Matrix Rigidity and FF Matrix Rigidity are W[1]-hard with respect to the parameter k. (The papers [17, 18] already imply that both of these problems are para-NP-hard with respect to the parameter r.) Our reduction is inspired by studies in Parameterized Complexity that involve the Odd Set problem [6], and consists of four reductions, one of which is builds upon the recent work of Bonnet et al. [2].

The necessity of having four different reductions stems from the fact that unlike previous studies of this nature, we establish the W[1]-hardness of our problem of interest over *any* finite field and over the field of reals rather than only over a specific finite field such as $\mathbb{F}_2$. Thus, we first need to define a special case of Odd Set, which we call Special Odd Set, and observe that its W[1]-hardness follows from the proof of the W[1]-hardness of Odd Set that is given in [6]. The correctness of our reductions crucially relies on the implications of the properties of this special case. Our first reduction translates Special Odd Set to a problem involving matrices rather than sets, which we call Special Odd Matrix. Then, to be able to discuss any finite field as well as the field of reals, we introduce new variants of Special Odd Matrix and the Nearest Codeword problem, called $\mathbb{F}$-Odd Matrix and $\mathbb{F}$-Nearest Codeword, respectively. The application of our second reduction results in an instance of $\mathbb{F}$-Odd Matrix. Then, the application of our third reduction, which builds upon [2], results in an instance of $\mathbb{F}$-Nearest Codeword. Finally, we devise a reduction whose application results in an instance of Matrix Rigidity. Here, we make explicit use of the fact that the rank of the target matrix can be large. The overall structure of the reduction may be relevant to studies of other problems where the field is not fixed.

2 Preliminaries

Due to space constraints, some standard notations and definitions have been omitted. The notation $[n]$ is used to denote the set of integers $\{1, \dots, n\}$. Given a matrix A of dimension $m \times n$, the i-th row of A is denoted by A_i, and the j-th column of A is denoted by A^j. The *rank* of a matrix is the cardinality of a maximum-sized collection of linearly independent columns (or rows), and is denoted by $\mathsf{rank}(A)$. We call a matrix $\tilde{A}$ a *jumbled matrix* of A if one can perform a series of row and column exchanges on $\tilde{A}$ to obtain the matrix A. Similarly, we call a matrix $\tilde{A}$ a *jumbled submatrix* of A if there exists a submatrix of A which is a jumbled matrix of $\tilde{A}$. A *mixed matrix* is a matrix having either an indeterminate or a value at each entry. We will be dealing with mixed matrices where the values belong to a finite field or $\mathbb{R}$. We use I_n to denote the *identity matrix* of size $n \times n$.

Let $x_1, \dots, x_n$ be variables. Then, a *monomial* is defined as a product $\prod_{i=1}^{n} x_i^{a_i}$ for non-negative integers $a_1, \cdots, a_n$. The degree of a variable x_i in a monomial $\prod_{i=1}^{n} x_i^{a_i}$ is defined to be the number a_i for $i \in [n]$. The degree of a *monomial* is defined as the sum of degrees of each variable occurring in it. A polynomial over a field $\mathbb{F}$ consists of a sum of monomials with coefficients from the field $\mathbb{F}$. The *total degree* of a polynomial is the degree of a monomial having maximum degree. A polynomial is referred to as *multi-linear* if the degree of any variable in it is at most 1. Given a system of polynomial equations $\mathcal{P} = \{P_1 = 0, P_2 = 0, \dots, P_m = 0\}$ over a field $\mathbb{F}$, we say that $\mathcal{P}$ is *feasible* over $\mathbb{F}$ if there exists an assignment of values from the field $\mathbb{F}$ to the variables in $\mathcal{P}$ which satisfies every polynomial contained in $\mathcal{P}$.

3 Dimension Reduction Procedure

In this section we show how to compress an input instance of Matrix Rigidity to an equivalent instance in which the matrix has at most $\mathcal{O}(r^2 \cdot k^2)$ entries. This is a crucial step in obtaining our FPT algorithms for Real Matrix Rigidity and FF Matrix Rigidity. In particular, this step will imply that FF Matrix Rigidity admits a polynomial kernel with respect to $r + k + p$.

Our algorithm is based on the following intuition. Suppose that A is a matrix of rank ℓ. If we can obtain a sequence $B_1, \dots, B_k$ of pairwise disjoint columns which form a basis of A, then the answer to the question "can we reduce the rank of A to a number $r < \ell$ by editing at most k entries in A" is completely determined by the answer to the same question where the editing operations are restricted to the submatrix of A formed by columns of $B_1, \dots, B_k$. However, ℓ could be much larger than $k + r$. Thus, to utilize this intuition to get the results we want, we need to do things slightly differently. For example, if the rank of the input matrix is too large, say at least $(k+1) \cdot (r+1)$, then we can partition the set of columns in a manner resulting in $k+1$ distinct submatrices of rank $r+1$ where, in order to reduce the rank of the input matrix to at most r, we must edit one entry in each of these submatrices. Thus, we can assume that the rank of the input matrix is upper bounded by $k \cdot (r+1)$, else we conclude that the input instance is a NO-instance. Then, the union of columns of a greedy collection of k disjoint subspaces of dimension at most p_i, where p_i is the maximum of $r+1$ and the rank of the matrix considered after deleting first $i-1$ selected subspaces, completely determines whether the rank of the input matrix can be reduced to at most r by editing at most k entries.

Now, let us move to the formal part of our arguments. Note that the relation 'is a *jumbled matrix* of' as defined in Section 2 is an equivalence relation. We need the following simple observations which follow from the definition of the rank of a matrix.

▶ **Observation 3.1.** *Let $A \in \mathbb{F}^{m \times n}$ be a matrix of rank equal to r. To make the rank of A at most $r-1$, one needs to change at least one entry in A.*

▶ **Observation 3.2.** *For a matrix A, let $\tilde{A}$ be a* jumbled matrix *of A. Then, the instances (A, r, k), (A^T, r, k), $(\tilde{A}, r, k)$ and $(\tilde{A}^T, r, k)$ are equivalent instances of* MATRIX RIGIDITY.

▶ **Observation 3.3.** *For a matrix A, let $\tilde{A}$ be its* jumbled submatrix. *Then $\mathsf{rank}(\tilde{A}) = \mathsf{rank}(A)$.*

Using Observation 3.3, we have the following.

▶ **Observation 3.4.** *If $\tilde{A}$ is a* jumbled submatrix *of A and $(\tilde{A}, r, k)$ is a NO-instance of* MATRIX RIGIDITY, *then (A, r, k) is also a NO-instance of* MATRIX RIGIDITY.

As stated before, our procedure greedily selects a subspace of appropriate dimension iteratively. A detailed description of the procedure, called COLUMN-REDUCTION, can be found in Figure 1. We will now explain the ideas necessary to understand this procedure, which is the heart of this section. The input to COLUMN-REDUCTION consists of a matrix A over any field, given along with non-negative numbers k and r. It outputs a matrix $\tilde{A}$ whose number of columns is bounded by a function of k and r such that the instances (A, r, k) and $(\tilde{A}, r, k)$ are equivalent instances of MATRIX RIGIDITY. The computation of a column basis and linearly independent vectors are done in the field $\mathbb{F}$ over which the matrix A is provided.

The procedure employs several variables. The variable i is used as an index variable whose initial value is 0, and it is incremented by 1 at a time. The case when the value of i exceeds k we will show that we are dealing with a NO-instance, otherwise the value depends on a particular input matrix A and is at most k. The variables $M_0, M_1, \ldots$ are submatrices of the input matrix A, satisfying the property that M_i is a submatrix of M_{i-1} with $M_0 = A$. In the first loop of COLUMN-REDUCTION (line 3), if the matrix M_i has rank at least $r+1$ then the variable L_i stores a set of $r+1$ linearly independent columns in the matrix M_i. Additionally, M_i can be obtained by appending the columns in L_i to the matrix M_{i+1}. The variable $i_{\leq r}$ is set to the value of i where the rank of M_i falls below $r+1$—after its initialization the value of$i_{\leq r}$ is not changed. In the second half of the procedure, similar to the set of variables L_i, we define a set of variables B_i which store a column basis of the matrix M_i (line 6). Recall that in this half of the procedure $i \geq i_{\leq r}$, and therefore each matrix M_i is of rank at most r. Additionally, M_i can be obtained by appending the columns in B_i to the matrix M_{i+1} for $i \geq i_{\leq r}$. Finally, the matrix $\mathcal{L}$ is constructed using all the columns in each matrix L_i, and the matrix $\mathcal{B}$ is constructed using all the columns in each matrix B_i for appropriate values of i. By Observation 3.1, we have to edit at least $i_{\leq r}$ entries of $\mathcal{L}$ to make its rank at most r.

In the procedure COLUMN-REDUCTION, a YES-instance of appropriate size can be obtained by taking the matrix $Z = [0]$ (of rank 0), which contains 0 as its only entry. Clearly, (Z, r, k) is a YES-instance of MATRIX RIGIDITY irrespective of the values of r and k. On the other hand, the instance (I_{r+k+1}, r, k) is a NO-instance of MATRIX RIGIDITY. Therefore, the matrix I_{r+k+1} can be used in place of a NO-instance of appropriate size. We need Z and I_{r+k+1} to satisfy the constraint that a kernel is an instance of the same problem as the input instance (even though, if the output is given by either line 1 or 4, we have actually solved the input instance (A, r, k) of MATRIX RIGIDITY in polynomial time). Using the procedure COLUMN-REDUCTION, it is straightforward to reduce the number of rows too. The details of this procedure are given in Figure 2.

▶ **Lemma 3.1.** *Let A be a matrix over some field $\mathbb{F}$, and let r and k be two non-negative integers. Given an instance (A, r, k), the procedure* MATRIX-REDUCTION *runs in time polynomial in input size and returns a matrix $\tilde{A}$ satisfying the following properties:*

```
Algorithm: Column-Reduction
Input: A matrix $A$ over some field $\mathbb{F}$, and two non-negative integers $r, k$.
Output: A matrix having $\mathcal{O}(rk)$ columns.

1. If $\mathsf{rank}(A) \leq r$ then return a YES-instance of appropriate size and exit.
2. Initialize $M_0 = A$ and $i = 0$.
3. While $\mathsf{rank}(M_i) \geq r + 1$:
    a. Let $L_i$ be a set of columns of $M_i$ which is linearly independent in $\mathbb{F}$ and whose
       size is $r + 1$.
    b. Let $M_{i+1}$ be the matrix obtained by deleting the columns in $L_i$ from $M_i$.
    c. Increment $i$ by 1.
4. If $i > k$ then return a NO-instance of appropriate size and exit.
   // The matrix $A$ has more than $k$ pairwise-disjoint blocks of the form $L_j$ for $j \leq i$,
   each having $r + 1$ linearly independent columns. By Observation 3.1, each block
   $L_i$ requires at least 1 edit, hence, by Observation 3.4, $(A, r, k)$ is a NO-instance of
   Matrix Rigidity.
5. Let $i_{\leq r} = i$ store the index where the rank of $M_i$ falls below $r + 1$.
6. While $i \leq k$:
    a. Let $B_i$ be a column basis of $M_i$.
    b. Obtain $M_{i+1}$ by deleting the columns in $B_i$ from $M_i$.
    c. If $M_{i+1}$ is empty (in other words, $B_i = M_i$) then return $A$.
    d. Increment $i$ by 1.
7. Let $\mathcal{L}$ be a matrix formed by the columns in each $L_i$ for $i \in \{0, \ldots, i_{\leq r} - 1\}$.
8. Let $\mathcal{B}$ be a matrix formed by the columns in each $B_i$ for $i \in \{i_{\leq r}, \ldots, k\}$.
9. Return the matrix formed by the columns in $\mathcal{L} \cup \mathcal{B}$.
```

Figure 1 The column reduction procedure.

1. *$\tilde{A}$ has at most $\mathcal{O}(r^2 \cdot k^2)$ entries.*
2. *If the output is produced by lines 6c and 9 of* Column-Reduction *(when called by* Matrix-Reduction*), then $\tilde{A}$ is a* jumbled submatrix *of A.*
3. *(A, r, k) is a YES-instance of* Matrix Rigidity *if and only if $(\tilde{A}, r, k)$ is a YES-instance.*

Proof. The steps of procedure Column-Reduction are all computable in polynomial time, and therefore Matrix-Reduction runs in polynomial time. We next turn to prove the desired properties. Let the matrix N' denote the output of Column-Reduction on the input instance (N, r, k).

Proof of 1: We first show that the size of the output is as claimed by procedure Column-Reduction. The output of this procedure is given at lines 1, 4, 6c and 9. If the output happens at line 1, it has 1 column by construction. Similarly, if the output happens at line 4, it has $r + k + 1 \leq (r + 1) \cdot (k + 1)$ columns by construction. If the output occurs at line 6c, then the number of columns in N' is at most $(k + 1) \cdot (r + 1)$ as it is constructed using columns of at most $i \leq k$ matrices, $L_0, \ldots, L_{i_{\leq r}-1}, B_{i_{\leq r}}, \ldots, B_i$, each having at most $r + 1$ columns. If the output happens at line 9, then $i = k$ and we get that N' has at most $(k + 1) \cdot (r + 1)$ columns.

The procedure Matrix-Reduction first obtains a matrix C_A with the aforementioned number of columns by running Column-Reduction. Then, it runs Column-Reduction again on the transpose of C_A to get its rows bounded. Thus, the dimensions of the output matrix are as claimed.

```
Algorithm: Matrix-Reduction
Input: A matrix $A$ over some field $\mathbb{F}$, and two non-negative integers $r, k$.
Output: A matrix having $\mathcal{O}(rk) \times \mathcal{O}(rk)$ enties.

1. Let $C_A$ = Column-Reduction$(A, r, k)$.
2. Let $R_A$ = Column-Reduction$(C_A^T, r, k)$.
3. Return $R_A^T$.
```

Figure 2 The dimension reduction procedure.

Proof of 2: The relation "is a jumbled submatrix of" is a transitive relation, therefore it suffices to show that the procedure Column-Reduction outputs a jumbled submatrix of A. If the output happens at lines 6c and 9, then the columns in the output matrix are a subset of the columns in the input matrix. Therefore, in the first line of procedure Matrix-Reduction C_A is a jumbled submatrix of A. Similarly, R_A is a jumbled submatrix of C_A^T. Finally note that for matrices X and Y, X is a jumbled submatrix of Y if and only if X^T is a jumbled submatrix of Y^T. Hence, the output matrix R_A^T is a jumbled submatrix of A.

Proof of 3: It suffices to show that the procedure Column-Reduction produces an equivalent instance of Matrix Rigidity. In the forward direction, suppose that (N, r, k) is a YES-instance of Matrix Rigidity. If the output occurs at line 1, it is a YES-instance by construction. The output cannot occur at line 4 as (N, r, k) is a YES-instance. At lines 6c and 9, by property 2, Column-Reduction outputs a *jumbled submatrix* N' of the input matrix N. Take any solution S, which consists of replaced entries and their indices, of the instance (N, r, k) of Matrix Rigidity, and denote the edited matrix obtained from N by N_S. Let S' be the list of entry indices in the matrix N' whose corresponding entries were edited by S in N. By Observation 3.3, $\mathsf{rank}(N'_{S'}) \leq \mathsf{rank}(N_S) \leq r$ (as $N'_{S'}$ is a jumbled submatrix of N_S), hence (N', r, k) is a YES-instance of Matrix Rigidity.

In the backward direction, suppose (N', r, k) is a YES-instance of Matrix Rigidity. If the output of N' occurs at lines 1 or 4, then we actually know the solution to the instance (N, r, k) of Matrix Rigidity (as explained in the pseudocode). If the output occurs at line 6c, then the output N' of Column-Reduction is a jumbled matrix of N and the result holds by Observation 3.2. Now we are left with the case when the output occurs at line 9. Let S' be any solution to the instance (N', r, k) of Matrix Rigidity, it consists of matrix indices and the value to be replaced. The matrix edited using a solution S' is denoted by $N'_{S'}$. Notice that the matrix N' consists of two submatrices $\mathcal{L}$ and $\mathcal{B}$. As $\mathcal{L}$ consists of $i_{\leq r}$ blocks having rank $r+1$, by Observation 3.1, we need to edit at least $i_{\leq r}$ entries in $\mathcal{L}$. So, we can afford to make at most $k - i_{\leq r}$ edits in the matrix $\mathcal{B}$. As $\mathcal{B}$ consists of $k + 1 - i_{\leq r}$ blocks, by pigeonhole principle there exists at least one block in $\mathcal{B}$, say B_t, which is not subject to any edit. The block B_t must have rank at most r, otherwise the solution will have left a block of rank more than r unedited implying that S' is not a solution. Notice that the block B_t spans the matrix M_{k+1}, which is obtained after deleting all the columns in the jumbled submatrix N' from N. Thus columns of the matrix M_{k+1} are in the span of the columns of $N'_{S'}$, which proves that the matrix N'' consisting of the columns of $N'_{S'}$ and M_{k+1} has rank at most r. We can easily construct a solution S for the matrix N by taking the same values corresponding to the entries edited by S' in N''. Note that N'' is a jumbled matrix of N_S, hence S is a solution for the instance (N, r, k) of Matrix Rigidity.

To complete the proof, observe that in the procedure Matrix-Reduction, the instances

Algorithm: MatRig-Alg
Input: A matrix A over a field $\mathbb{F}$, and two non-negative numbers r, k.
Output: Can one edit at most k entries of A to obtain a matrix of rank at most r?

```
1. Let $A'$ =Matrix-Reduction$(A, r, k)$.
2. For each set $E$ of $k$ entries in $A'$:
   a. Replace each entry of $A'$ indexed by an element in $E$ by a distinct indeterminate
      to obtain a mixed matrix $A'_E$.
   b. Let $\mathcal{P}$ be the set of equations obtained by setting the determinant of each $(r +
      1) \times (r + 1)$ submatrix of $A'_E$ to 0.
   c. If $\mathcal{P}$ is feasible over $\mathbb{F}$ then return YES and exit.
3. Return NO and exit.
```

Figure 3 Description of the algorithm for Matrix Rigidity.

(A, r, k) and (C_A, r, k) are equivalent by the argument above. By Observation 3.2, (C_A, r, k) and (C_A^T, r, k) are equivalent. As R_A is the output of Column-Reduction,(R_A, r, k) is equivalent to (C_A, r, k). Finally, by Observation 3.2, again (R_A, r, k) and (R_A^T, r, k) are equivalent. ◀

If the matrix A is over a fixed finite field $\mathbb{F}$, we obtain a kernel as well.

▶ **Theorem 3.1.** *Given an instance (A, r, k) of* FF Matrix Rigidity *over the field $\mathbb{F}_p$, the procedure* Matrix-Reduction *outputs an $\mathcal{O}(r^2 \cdot k^2 \cdot \log p)$-kernel.*

Proof. The number of entries in the output matrix of Matrix-Reduction is bounded by $\mathcal{O}(r^2 \cdot k^2)$, and the bit length of each entry is at most $\lceil \log_2 p \rceil$. ◀

In case the field $\mathbb{F}$ is infinite—for example, if $\mathbb{F}$ is either $\mathbb{Q}$ or $\mathbb{R}$—the procedure is not guaranteed to produce a kernel as the bit lengths of matrix entries may not be bounded by a function of r and k.

4 Fixed-Parameter Tractability with Respect to $k + r$

This section presents an algorithm for Matrix Rigidity. Using Lemma 3.1, we can reduce any instance (A, r, k) to an equivalent instance (A', r, k) such that the matrix A' is a jumbled submatrix of A and the number of entries in A' is $\mathcal{O}(r^2 \cdot k^2)$. Once we have such a matrix A', it is useful to examine an alternative definition of the rank of a matrix, which is given in terms of the determinant of its square submatrices. Specifically, we will rely on the following lemma.

▶ **Lemma 4.1** (Chapter 7 [23]). *A matrix A over $\mathbb{R}$ has rank at most r if and only if all the $(r + 1) \times (r + 1)$ submatrices of A have determinant 0.*

The correctness of our algorithm MatRig-Alg for Matrix Rigidity, which is described in Figure 3, follows in a straightforward fashion using Lemma 4.1. This algorithm for Matrix Rigidity crucially relies on a procedure which can decide the feasibility of a system of polynomials over a given field. This procedure shall be the object of discussion in the rest of the section.

Observe that each polynomial in $\mathcal{P}$, as defined in the algorithm MatRig-Alg, has at most k unknowns and its *total degree* is at most k. The size of $\mathcal{P}$ is of order $(r \cdot k)^{\mathcal{O}(r)}$. In the case where the underlying field is $\mathbb{R}$, suppose the longest length entry has bit length L. The coefficients of polynomials in $\mathcal{P}$ are obtained by computing the determinant of matrices which have size at most $r \times r$. By *Hadamard's inequality* [12], for a matrix M of size $r \times r$ the $\det(M) \leq \prod_{i \in [r]} \|M_i\|$. If the bit length of entries in M is at most L then the coefficients of polynomials in $\mathcal{P}$ can be shown to be bounded by $L' = r \cdot L + \mathcal{O}(r \cdot \log r)$.

We use the following proposition, given in [1] (Proposition 13.19), to check the feasibility of the system of polynomials $\mathcal{P}$ when it is defined over $\mathbb{R}$.

▶ **Proposition 2.** *Given a set $\mathcal{P}$ of m polynomials of degree d in k variables with coefficients in C, we can decide with complexity $m \cdot d^{\mathcal{O}(k)}$ in D (where D is the ring generated by the real and imaginary parts of the coefficients of the polynomials in $\mathcal{P}$) whether* $\mathrm{Zer}(\mathcal{P}, C^k)$ *is empty. Moreover, if $D = \mathbb{Z}$ and the bitsizes of the coefficients of the polynomials are bounded by τ, then bitsizes of the integers appearing in the intermediate computations and the output are bounded by $(\tau + \log m) \cdot d^{\mathcal{O}(k)}$.*

Applying the proposition above on the system of equations $\mathcal{P}$, we get the following.

▶ **Theorem 4.1.** *Suppose we are given a matrix A over $\mathbb{R}$ such that the bit length of each of its entries is bounded by L, and let r and k be two non-negative integers. Then, the instance (A, r, k) of* REAL MATRIX RIGIDITY *can be solved in time $\mathcal{O}^*(2^{\mathcal{O}((r+k)\cdot\log(r\cdot k))})$. Here, the bit lengths of integers appearing in intermediate computations and the output are bounded by $\mathcal{O}(r \cdot (\log(k \cdot r) + L)) \cdot 2^{\mathcal{O}(k \cdot \log k)}$.*

Proof. The algorithm MATRIG-ALG generates $\mathcal{O}((r \cdot k)^{2k})$ many systems of equations. Each system of equations has $m = (r \cdot k)^{\mathcal{O}(r)}$ many equations, where the degree $d = k$ and there are k variables. Using Proposition 2, we get the required running time and the bit lengths.

Notice that a system of equations $\mathcal{P}$ is feasible if and only if the chosen entries of the matrix can be edited to reduce the rank. Since we exhaustively try all possible entries that can be edited, the correctness of MATRIG-ALG follows. ◀

In the case where the underlying field $\mathbb{F}_p$ is finite, the coefficients of the polynomials are elements of $\mathbb{F}_p$ and hence have bounded bit lengths. The feasibility of $\mathcal{P}$ over a finite field can be decided using the following known algorithm which also gives us an algorithm for FF MATRIX RIGIDITY.

▶ **Proposition 3** ([10]). *There is a deterministic algorithm which, given an input consisting of a finite field $\mathbb{F}_p$ and system of polynomials $f_1, \cdots, f_m \in \mathbb{F}_p[x_1, \cdots, x_k]$ of* total degree *bounded by d, decides the feasibility of the system in time $d^{k^{\mathcal{O}(k)}} \cdot (m \cdot \log p)^{\mathcal{O}(1)}$.*

Similar to the proof of Theorem 4.1, we obtain the following.

▶ **Theorem 4.2.** *The problem* FF MATRIX RIGIDITY*, where the input matrix A is an $m \times n$ matrix over a field $\mathbb{F}_p$, can be solved in time $f(r, k)(\log p + m + n)^{\mathcal{O}(1)}$ for some function f.*

This algorithm for FF MATRIX RIGIDITY has the advantage that it runs in time which is *polynomial in the logarithm of the order of the field*, even though the dependence on k is exponential.

5 W[1]-Hardness with Respect to k

In this section, we first reduce (in two steps) a special case of ODD SET to a problem that has a formulation easier to use in our context. The latter problem is reduced to a variant of NEAREST CODEWORD, which, in its turn, is reduced to REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY.

▶ **Theorem 4.** Real Matrix Rigidity *and* FF Matrix Rigidity *for any choice of a finite field $\mathbb{F}_p$ are $W[1]$-hard with respect to k.*

Proof. Let $\mathbb{F}$ denote the field, which is either $\mathbb{R}$ or some finite field $\mathbb{F}_p$, over which we define Matrix Rigidity. First, we observe that the reduction from Multicolored Clique given in the book [3] (Theorem 13.31) to show that Odd Set is W[1]-hard actually shows that the following special case of Odd Set is W[1]-hard. That is, the constructed instances have the form specified in the special case.

Special Odd Set **Parameter:** k
Input: A family $\mathcal{F}$ of sets over a universe U, a non-negative integer k, a partition $(U_1, \ldots, U_k)$ of U such that for every $i \in [k]$, $U_i \in \mathcal{F}$, and for every $F \in \mathcal{F}$, there exist $i, j \in [k]$ for which $F \subseteq U_i \cup U_j$.
Question: Is there a subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has size 1?

The arguments below will crucially rely on the fact that we restrict ourselves to this special case. Given a vector v, we let $\bar{I}_0(v)$ denote the indices of the entries of v that do not contain 0. Now, we reformulate Special Odd Set in the language of matrices as follows.

Special Odd Matrix **Parameter:** k
Input: A $t \times r$ binary matrix L over $\mathbb{R}$, a non-negative integer k, a partition $(U_1, \ldots, U_k)$ of $[r]$ such that for every $i \in [k]$, there exists $j \in [t]$ for which $U_i = \bar{I}_0(L_j)$, and for every $i \in [t]$, there exist $j, \ell \in [k]$ for which $\bar{I}_0(L_i) \subseteq U_j \cup U_\ell$.
Question: Is there an r-dimensional binary vector x such that $|\bar{I}_0(x)| \leq k$ and $Lx = 1$?

Given an instance $(\mathcal{F}, U, (U_1, \ldots, U_k), k)$ of Special Odd Set, it is straightforward to obtain (in polynomial time) an equivalent instance $(L_{t\times r}, (U'_1, \ldots, U'_k), k')$ of Special Odd Matrix as follows. First, we let $t = |\mathcal{F}|$ and $r = |U|$. We assume w.l.o.g. that $U = [r]$. Now, we associate a row L_i with each set $F \in \mathcal{F}$ by letting L_i contain 1 at each entry whose index belongs to F and 0 at each of the remaining entries. That is, $\bar{I}_0(L_i) = F$. Finally, we let $(U'_1, \ldots, U'_k) = (U_1, \ldots, U_k)$ and $k' = k$. It is easy to see that $S \subseteq U$ is a solution to $(\mathcal{F}, U, (U_1, \ldots, U_k), k)$ if and only if the binary vector $x_{r\times 1}$ such that $\bar{I}_0(x) = S$ is a solution to $(L_{t\times r}, (U'_1, \ldots, U'_k), k)$, and therefore the instances are equivalent.

We now incorporate the input field $\mathbb{F}$.

$\mathbb{F}$-Odd Matrix **Parameter:** k
Input: A $t \times r$ binary matrix L over $\mathbb{F}$ and a non-negative integer k.
Question: Is there an r-dimensional vector x over $\mathbb{F}$ such that $|\bar{I}_0(x)| \leq k$ and $Lx = 1$?

We reduce Special Odd Matrix to $\mathbb{F}$-Odd Matrix as follows.Given an instance $(L_{t\times r}, (U_1, \ldots, U_k), k)$ of Special Odd Matrix, we simply output $(L_{t\times r}, k)$ as the equivalent instance of $\mathbb{F}$-Odd Matrix. In one direction, let x be a solution to $(L_{t\times r}, (U_1, \ldots, U_k), k)$. Recall that L is a binary matrix. Thus, since x is a binary vector satisfying $Lx = 1$ over $\mathbb{R}$, it must also satisfy $Lx = 1$ over $\mathbb{F}$. Since $|\bar{I}_0(x)| \leq k$, we get that x is a solution to $(L_{t\times r}, k)$. In the second direction, let x be a solution to $(L_{t\times r}, k)$. Since $|\bar{I}_0(x)| \leq k$ and for every $s \in [k]$, there exists $s' \in [t]$ for which $U_s = \bar{I}_0(L_{s'})$, it must hold that for every $s \in [k]$, $|\bar{I}_0(x) \cap U_{s'}| = 1$. Thus, since for every $s \in [k]$, $L_{s'}x = 1$ over $\mathbb{F}$, it holds that x is a binary vector. It remains to show that $Lx = 1$ over $\mathbb{R}$. Consider some index $i \in [t]$. Then, there exist $j, \ell \in [k]$ such that $\bar{I}_0(L_i) \subseteq U_j \cup U_\ell$. Therefore, since for every $s \in [k]$, $|\bar{I}_0(x) \cap U_s| = 1$, and since both L and x are binary, it is only possible that $L_i x = 1$ over $\mathbb{F}$ if

$|\bar{I}_0(x) \cap \bar{I}_0(L_i)| = 1$, which allows us to conclude that $L_i x = 1$ also over $\mathbb{R}$.

In what follows, calculations are performed over $\mathbb{F}$. Next, we reduce $\mathbb{F}$-Odd Matrix to the following variant of the Nearest Codeword problem. This specific reduction is inspired by a reduction from Nearest Codeword to Odd Set of Bonnet et al. [2].

$\mathbb{F}$-Nearest Codeword **Parameter:** k
Input: An $m \times n$ matrix M, an m-dimensional vector b over $\mathbb{F}$, and a non-negative integer k.
Question: Is there an n-dimensional vector y over $\mathbb{F}$ such that the Hamming distance between My and b is at most k?

Given an instance $(L_{t\times r}, k)$ of $\mathbb{F}$-Odd Matrix, we construct an instance $(M_{m\times n}, b, k')$ of $\mathbb{F}$-Nearest Codeword as follows. First, let $k' = k$. Now, let M be an $m \times n$ matrix, where $m = r$ and $n = r - \mathsf{rank}(L)$, such that the rows of L form a basis for the subspace perpendicular to the column space of M. Then, an r-dimensional vector v over $\mathbb{F}$ satisfies $Lv = 0$ if and only if v belongs to the column space of M (i.e., there is an n-dimensional vector y over $\mathbb{F}$ such that $My = v$). Finally, let b be an r-dimensional vector such that $Lb = -1$. If no such vector exists, then there is also no r-dimensional vector over $\mathbb{F}$ such that $Lv = 1$, which in particular implies that $(L_{t\times r}, k)$ is a NO-instance, and thus we can return a trivial NO-instance of $\mathbb{F}$-Nearest Codeword. Therefore, next assume that b exists. To prove that the reduction is correct, first let x be a solution to $(L_{t\times r}, k)$. Then, $Lx = 1$, and since $Lb = -1$, we have that $L(x + b) = Lx + Lb = Lx - 1 = 0$. Therefore, by the choice of M, there exists an n-dimensional vector y over $\mathbb{F}$ such that $My = (x + b)$. Since $|\bar{I}_0(x)| \leq k$, we have that the Hamming distance between My and b is at most k, which implies that y is a solution to $(M_{m\times n}, b, k')$. In the other direction, let y be a solution to $(M_{m\times n}, b, k')$. Then, since the Hamming distance between My and b is at most k, there exists an m-dimensional vector x such that $|\bar{I}_0(x)| \leq k$ and $My = x + b$. Therefore, by the choice of M, $L(x + b) = 0$. Since $Lb = -1$, we get that $Lx = 1$, which implies that x is a solution to $(L_{t\times r}, k)$.

Finally, we reduce $\mathbb{F}$-Nearest Codeword to Matrix Rigidity over $\mathbb{F}$. For this purpose, let $(M_{m\times n}, b, k)$ be an instance of $\mathbb{F}$-Nearest Codeword. We assume w.l.o.g. that the columns of M are linearly independent. We construct an equivalent instance $(A_{a\times b}, r, k')$ of Matrix Rigidity over $\mathbb{F}$ as follows. First, let $k' = k$. Now, let $a = m$, $r = n$ and $b = (k + 1)n + 1$. For each $i \in [k + 1]$ and $j \in [n]$, we define $A^{(k+1)(i-1)+j} = M^j$. Finally, we define $A^{(k+1)m+1} = b$. On the one hand, let y be a solution to $(M_{m\times n}, b, k)$. Then, there are at most k entries that should be changed in b to obtain an m-dimensional vector b' over $\mathbb{F}$ such that $My = b'$. In the matrix A, replace the last column b by b'. Denote the resulting matrix by A'. Then, the last column of A' is a linear combination of its other columns (by the construction of A and since $My = b'$), and among the other columns of A', there are only m distinct columns. Therefore, $\mathsf{rank}(A') = n$, which implies that $(A_{a\times b}, r, k')$ is a YES-instance. In the other direction, suppose that $(A_{a\times b}, r, k')$ is a YES-instance. Then, it is possible to change at most k entries in A and obtain a matrix A' such that $\mathsf{rank}(A') = n$. Since besides the last column of A, each column of A is repeated $k + 1$ times (i.e., more times than the number of changes), and there are n such distinct rows, it must be that the last column of A' is a linear combination of the distinct columns of A excluding the last column of A. By the construction of A, we get that there exists an n-dimensional vector y over $\mathbb{F}$ such that $My = b'$, where b' is the last column of A'. Since the Hamming distance between b and b' is at most k, we have that y is a solution to $(M_{m\times n}, b, k)$. ◀

References

1 Basu, S., Pollack, R., Roy, M.F.: Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

2 Bonnet, E., Egri, L., Marx, D.: Fixed-parameter approximability of boolean MinCSPs. In: ESA. pp. 18:1–18:18 (2016)

3 Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized algorithms. Springer (2015)

4 Damaschke, P., Mogren, O.: Editing simple graphs. In: WALCOM. pp. 249–260 (2014)

5 Deshpande, A.J.: Sampling-based algorithms for dimension reduction. Ph.D. thesis, Massachusetts Institute of Technology (2007)

6 Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer (2013)

7 Friedman, J.: A note on matrix rigidity. Combinatorica 13(2), 235–239 (1993)

8 Grigoriev, D.: Using the notions of separability and independence for proving the lower bounds on the circuit complexity (in russian). Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka (1976)

9 Grigoriev, D.: Using the notions of separability and independence for proving the lower bounds on the circuit complexity. Journal of Soviet Math. 14(5), 1450–1456 (1980)

10 Kayal, N.: Solvability of a system of bivariate polynomial equations over a finite field. In: ICALP. pp. 551–562 (2005)

11 Kumar, A., Lokam, S.V., Patankar, V.M., Sarma, M.N.J.: Using elimination theory to construct rigid matrices. Computational Complexity 23(4), 531–563 (2013)

12 Lange, K.: Hadamard's determinant inequality. The American Mathematical Monthly 121(3), 258–259 (2014)

13 Lokam, S.V.: Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity. In: FOCS. pp. 6–15 (1995)

14 Lokam, S.V.: On the rigidity of Vandermonde matrices. Theoretical Computer Science 237(1–2), 477 – 483 (2000)

15 Lokam, S.V.: Complexity lower bounds using linear algebra. Found. Trends Theor. Comput. Sci. 4, 1–155 (Jan 2009)

16 Mahajan, M., Sarma M.N., J.: On the complexity of matrix rank and rigidity. In: CSR. pp. 269–280 (2007)

17 Meesum, S.M., Misra, P., Saurabh, S.: Reducing rank of the adjacency matrix by graph modification. In: COCOON. pp. 361–373 (2015)

18 Meesum, S.M., Saurabh, S.: Rank reduction of directed graphs by vertex and edge deletions. In: LATIN. pp. 619–633 (2016)

19 Murota, K.: Matrices and matroids for systems analysis. Springer (2000)

20 Razborov, A.A.: On rigid matrices. Manuscript in russian (1989)

21 Sarma M.N., J.: Complexity Theoretic Aspects of Rank, Rigidity and Circuit Evaluation. Ph.D. thesis, The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai (2009)

22 Shokrollahi, M.A., Spielman, D., Stemann, V.: A remark on matrix rigidity. Information Processing Letters 64(6), 283 – 285 (1997)

23 Sigler, L.: Algebra. Undergraduate Texts in Mathematics, Springer-Verlag (1976)

24 Valiant, L.G.: Graph-theoretic arguments in low-level complexity. In: MFCS. pp. 162–176 (1977)

A Detailed Preliminaries

The notation $[n]$ is used to denote the set of integers $\{1, \ldots, n\}$. The *Hamming distance* between two strings of equal length is the number of positions at which they differ.

Linear Algebra: The symbols $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{F}_p$ are used to denote the field of real numbers, the field of rational numbers, and a finite field of order p, respectively. We also use the unsubscripted symbol $\mathbb{F}$ to denote a field, in which case its order is denoted by $|\mathbb{F}|$. A vector v of length n is an ordered tuple of n values from a field $\mathbb{F}$. A collection of vectors $\{v_1, v_2, \ldots, v_k\}$ is said to be *linearly dependent* if there exist values $a_1, a_2, \ldots, a_k \in \mathbb{F}$, not all equal 0, such that $\sum_{i=1}^{k} a_i v_i = 0$; otherwise the collection is said to be *linearly independent*.

A matrix A of dimension $m \times n$ is a sequence of values (a_{ij}) for $i \in [m]$ and $j \in [n]$. The i-th row of A, denoted by A_i, is defined as the vector $(a_{i1}, a_{i2}, \ldots, a_{in})$, and the j-th column of A, denoted by A^j, is defined as the vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$. Given $I \subseteq [m]$ and $J \subseteq [n]$, we define $A[I, J] = \big(a_{ij} \ : \ i \in I, \ j \in J\big)$, i.e., $A[I, J]$ is the submatrix (or minor) of A with the row set I and the column set J. We call a matrix A' a *jumbled matrix* of A if one can perform a series of row and column exchanges on A' to obtain the matrix A. Similarly, we call a matrix A' a *jumbled submatrix* of A if there exists a submatrix of A which is a jumbled matrix of A'. For a matrix A (or a vector v) by A^T (or v^T) we denote the *transpose*. The *rank* of a matrix is the cardinality of a maximum-sized collection of linearly independent columns (or rows), and is denoted by $\mathsf{rank}(A)$. We use I_n to denote the *identity matrix* of size $n \times n$. A *mixed matrix* is a matrix having either an indeterminate or a value at each entry. We would be dealing with mixed matrices where the values belong to a finite field or $\mathbb{R}$. A *completion matrix* is obtained by substituting indeterminates in a *mixed matrix* with values from a finite field or $\mathbb{R}$.

Polynomials: Let $x_1, \ldots, x_n$ be variables. Then, a *monomial* is defined as a product $\prod_{i=1}^{n} x_i^{a_i}$ for non-negative integers $a_1, \cdots, a_n$. The degree of a variable x_i in a monomial $\prod_{i=1}^{n} x_i^{a_i}$ is defined to be the number a_i for $i \in [n]$. The degree of a *monomial* is defined as the sum of degrees of each variable occurring in it. A polynomial over a field $\mathbb{F}$ consists of a sum of monomials with coefficients from the field $\mathbb{F}$. The *total degree* of a polynomial is the degree of a monomial having maximum degree. A polynomial is referred to as *multi-linear* if the degree of any variable in it is at most 1. Given a system of polynomial equations $\mathcal{P} = \{P_1 = 0, P_2 = 0, \ldots, P_m = 0\}$ over a field $\mathbb{F}$, we say that $\mathcal{P}$ is *feasible* over $\mathbb{F}$ if there exists an assignment of values from the field $\mathbb{F}$ to the variables in $\mathcal{P}$ which satisfies every polynomial contained in $\mathcal{P}$.

B Proof of Lemma 4.1

Define the length of an $n \times n$ matrix to be n. The rank of a matrix is equal to the length of its largest square submatrix having non-zero determinant. Thus, the forward direction follows from this definition.

For the backward implication, assume to the contrary that the rank of A is strictly more than r. Therefore, there is a square submatrix B of A whose length is strictly more than r and whose determinant is not 0. For any $m \times m$ matrix B, by the Laplace expansion of determinants [[19], Proposition 2.1.3], it holds that

$$\det(B) = \sum_{J \subseteq [m], |J| = |I|} (-1)^{\sum I + \sum J} \det(B_{I,J}) \det(B_{\overline{I},\overline{J}}) \tag{1}$$

where $I \subseteq [m]$, $\bar{I} = [m] \setminus I$ and $\sum I = \sum_{i \in I} i$. As B is a submatrix of A, all its $(r+1) \times (r+1)$ submatrices have determinant 0 by assumption. In equation 1, let $I \subseteq [m]$ be a set with $|I| = r + 1$—with this setting, it is easy to see that $\det(\mathrm{B}) = 0$, and thus we obtain a contradiction. Hence, all the submatrices of A with length at least $r + 1$ have determinant 0, implying that the rank of A is at most r. ◀

C The Definitions of Odd Set and Nearest Codeword

Odd Set **Parameter:** k
Input: A family $\mathcal{F}$ of sets over a universe U and a non-negative integer k.
Question: Does there exist a subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has odd size?

Nearest Codeword **Parameter:** k
Input: An $m \times n$ matrix M and an m-dimensional vector b over $\mathbb{F}_2$, along with a non-negative integer k.
Question: Is there an n-dimensional vector x over $\mathbb{F}_2$ such that the Hamming distance between Mx and b is at most k?

D An Algorithm for FF Matrix Rigidity with Subexponential Dependency on k

In this section, we will also rely on the classic technique of bounded search trees, which is presented in Appendix E. Our objective is to prove the following theorem.

▶ **Theorem 5.** *The* FF Matrix Rigidity *problem is solvable in time* $\mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$ *for some function* f.

Let $\mathbb{F}_p$ be a finite field. We will prove that Matrix Rigidity over $\mathbb{F}_p$ is solvable in the desired time. Let $(A_{m\times n}, r, k)$ be an instance of this problem. Meesum et al. [18] proved that any rank r skew-symmetric matrix A with entries from $\{-1, 0, 1\}$ has at most 3^r distinct columns (Theorem 2 in [18]). Their proof (with a straightforward modification) actually shows that any rank r symmetric matrix A with entries from $\mathbb{F}_p$ has at most p^r distinct rows and at most p^r distinct columns.

Given a matrix M, let I be a maximum sized set of distinct rows of M, and let J be a maximum sized set of distinct columns of M. Then, we define $\mathsf{distinct}(M) = M[I, J]$. To be more precise, $\mathsf{distinct}(M)$ should be defined as an equivalence class of submatrices up to reordering of rows and columns, but here we slightly abuse notation and consider some specific submatrix $M[I, J]$ as $\mathsf{distinct}(M)$. Now, let $\mathcal{D}$ be the set of all matrices with distinct rows and distinct columns whose rank is r, and which have at most p^r rows as well as at most p^r columns. Observe that $|\mathcal{D}|$ is bounded by a function that depends only on r and p. Therefore, to solve $(A_{m\times n}, r, k)$ in the desired time, it is sufficient, for each $D \in \mathcal{D}$, to check in time $\mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$ for some function f whether it is possible to change at most k entries in A to obtain a matrix M such that $\mathsf{distinct}(M) = D$.

Now, suppose that we are given a matrix $D \in \mathcal{D}$. Given a matrix M, we let $\mathsf{sym}(M)$ denote the matrix $\begin{bmatrix} 0 & M \\ M^{\mathrm{T}} & 0 \end{bmatrix}$, where 0 is the matrix of appropriate dimension that contains

0 at each of its entries. Observe that at most k entries in A can be changed to obtain a matrix M such that $\mathsf{distinct}(M) = D$ if and only if there are at most k pairs (i, j), $1 \le i \le m$ and $1 \le j \le n$, such that the entries $a_{i,j+m}$ and $a_{i+n,j}$ in $\mathsf{sym}(A)$ can be changed to obtain a matrix M such that $\mathsf{distinct}(M) = \mathsf{sym}(D)$. Now, we think of the matrices $\mathsf{sym}(A)$ and $\mathsf{sym}(D)$ as adjacency matrices of weighted undirected complete graphs where the weights belong to $\mathbb{F}_p$. More precisely, given a symmetric matrix $M_{t\times t}$ with zeros at its diagonal, the construction of $\mathsf{graph}(M)$ is performed as follows. For each index $i \in [t]$ we introduce a vertex v_i, and the weight of an edge $\{v_i, v_j\}$ is given at the entry m_{ij}. Given a weighted undirected complete graph $G = (V, E)$, let $(V_1, \ldots, V_\ell)$ be a partition of V minimizing ℓ such that for all $i \in [l]$, the weight of each edge between any two vertices in V_i is 0, and for all distinct $i, j \in [l]$, $v, v' \in V_i$ and $u \in V_j$, the weight of $\{v, u\}$ equals the weight of $\{v', u\}$. Observe that this partition is unique up to reordering the sets V_i. Now, we let $\mathsf{distinct}(G)$ be the weighted undirected complete graph having one vertex representing each set V_i, and letting the edge between the vertex representing V_i and the vertex representing V_j have the same weight as any edge between a vertex in V_i and a vertex in V_j in G. Since changing an entry in a matrix M is equivalent to changing the weight of an edge in $\mathsf{graph}(\mathsf{sym}(M))$, we conclude that to prove Theorem 5, it is sufficient to prove the following lemma.

▶ **Lemma 6.** *Let G and H be weighted undirected complete graphs with weights from $\mathbb{F}_p$ such that $\mathsf{distinct}(H) = H$ and $|V(H)| = g(r, p)$ for some function g. Then, it is possible to determine in time $\mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$ for some function f whether the weights of at most k edges in G can be changed to obtain a graph G' such that $\mathsf{distinct}(G') = H$.*

Thus, in the rest of this section, it is sufficient to focus on the proof of Lemma 6, which is based on a proof given in the paper [18] (which, in turn, is inspired by the paper [4]). To this end, we will also need the following result, whose proof can be found in [4].

▶ **Proposition 7.** *The branching vector $(1, t, t, \ldots, t)$ for some $t > 0$ that appears s times has branching number bounded by $1 + \dfrac{\log_2 t}{t}$ if t is significantly larger than s.*

Proof of Lemma 6. A vertex of H will be referred to as a bag and will be filled in with the vertices of G. Let $b = |V(H)|$ denote the number of bags in H and use $\mathcal{B} = \{B_i : i \in [b]\}$ to denote the set of bags. Assume that the vertices in H are $v_1, \ldots, v_b$ and B_i corresponds to the vertex v_i, for all $i \in [b]$. The collection of vertex-sets of G corresponding to the vertices in $\mathsf{distinct}(G)$ is denoted by $\mathcal{M} = \{V_1, \ldots, V_t\}$ (each V_i denotes a different set of vertices in G represented by the same vertex in $\mathsf{distinct}(G)$). Observe that each change of a weight of an edge in G can decrease the number of vertices in $\mathsf{distinct}(G)$ by at most 2, and therefore if $t > 2k + |V(H)|$, the answer to (G, H, r, p, k) is NO. Therefore, we may next assume that $t \le 2k + |V(H)|$. If at any point during the calculations below, the value of the parameter k drops below 0, we return the answer NO at the current node of the search tree; if at least one leaf of the search tree returns YES, we propagate the value YES – that is, if a node has several children (corresponding to different branches considered by a branching rule) and at least one of them returns YES, we return YES. In the preprocessing phase of the algorithm we add a “sufficient” number of vertices to each bag, which will allow us later to perform branching rules associated with branching vectors where the drop in the parameter, at all but one branch, is large.

Preprocessing phase: Each bag can be in one of two states: *closed* or *open*. At the start of the algorithm all the bags are empty and open. In every bag we create $s = \lfloor\sqrt{k}\rfloor$ slots. Overall there are $b \cdot s$ slots that can be filled. We say that a bag is *free* if it is open and less than s slots have been filled in it. The state of a node in the bounded search tree is denoted using $(\mathcal{M}, \mathcal{B})$. As long as there is a free bag, we perform the following branching rule, consisting of $t+1$ branches. In its i^{th} branch, for $i \in [t]$, pick an arbitrary vertex u from V_i, delete it from V_i and add it to the lowest number free bag in $\mathcal{B}$. In the $(t+1)^{th}$ branch we mark the lowest number free bag as closed. The application of branching rules in the preprocessing phase is finished when no bag is free. At each leaf node $(\mathcal{M}, \mathcal{B})$ of the bounded search tree, we construct a graph H' over the vertices in $\bigcup \mathcal{B}$. For every weighted edge $\{v_i, v_j\}$ in H we add all the edges $B_i \times B_j$ in the graph H' with the same weight as $\{v_i, v_j\}$. Edges between vertices of the same bag have weight 0. Next for every edge $\{v, u\}$ in H', we check whether the weight of the edge in H' is the same as its weight in G – if this is not the case, we decrease k by 1. This operation is equivalent to changing the weight of the edge in G to its weight in H'.

At the end of the preprocessing phase, for each vertex v_i of H we either know exactly which are the at most s vertices of G that should be equivalent to it in the graph G' we need to obtain as a solution by making at most k changes of edge-weights in G (i.e., $\mathsf{distinct}(G') = H$), or we know exactly s vertices of G equivalent to v_i in G'. In the first case, the bag has been closed, and in the second case, it remains open. In the preprocessing phase, each branching rule consists of at most $t+1$ branches and the depth of the bounded search tree is $b \cdot s$. Therefore, this phase can be performed in time $\mathcal{O}^*((t+1)^{b \cdot s}) = \mathcal{O}^*((b+2k+1)^{b \cdot s}) = \mathcal{O}^*(2^{\mathcal{O}(f'(r,p)\sqrt{k}\log k)})$ for some function f'.

Assigning bags to undecided vertices: This phase of the algorithm begins at a node $(\mathcal{M}, \mathcal{B})$ along with the graph H' and the reduced parameter k as provided by a leaf of the bounded search tree procedure in the previous phase. The vertices in $V(G) \setminus (\bigcup \mathcal{B})$, which have not yet been added to any bag, are called *undecided vertices*. We note that so far the weight modifications have been done only within the bag vertices added in the preprocessing phase, and that the (possibly modified) weights of edges between these bag vertices remain fixed for the rest of the algorithm. The branching rules are applied exhaustively in the given order – if at any node of the search tree a rule is applied, then none of the previous rules is applicable. We first consider the case when all the bags are open and and handle the closed bags later.

If there exists an undecided vertex u and a bag $B_i \in \mathcal{B}$ such that not all of the edges between u and the vertices in B_i have the same weight then we apply the following rule. For each weight in $\mathbb{F}_p$, we have a separate branch. In the branch corresponding to some weight $w \in \mathbb{F}_p$, for each edge between u and a vertex in B_i whose weight is not w, we change the weight of the edge to w and decrease k by 1. For now, u is not yet added to any bag, and remains an undecided vertex. The branching vector obtained is at least as good as the ones corresponding to the following two extreme cases. In the first case, all of the edges but one between u and the vertices of B_i have the same weight w. Then, at the branch corresponding to w the parameter is decreased by 1, at the branch corresponding to the weight which is not w of an edge between u and a vertex in B_i the parameter is decreased by $s-1$, and at each of the other $p-2$ branches the parameter is decreased by s. This corresponds to the branching vector $(1, s-1, s, \dots, s)$, where s appears $p-2$ times, whose branching number, by Proposition 7, is bounded by $1 + \dfrac{\log_2 s}{s}$. In the second case, for each weight in $\mathbb{F}_p$, about s/p of the edges between u and the vertices in B_i have this weight. Then, we obtain the branching vector $((p-1)s/p, \dots, (p-1)s/p)$, where $(p-1)s/p$ appears p times, whose branching number is the solution to $x^{\frac{(p-1)s}{p}} = p$, which is $p^{\frac{p}{(p-1)s}}$. Now, after applying this rule exhaustively, given any undecided vertex u, for each bag in $\mathcal{B}$ the weights of the edges between u and the vertices in this bag are the same.

We say that a vertex u fits a bag $B(u) \in \mathcal{B}$, if u has edges of the same weights as vertices in $B(u)$ in the graph induced on the bags. That is, the weight of each edge between u and a vertex in $B(u)$ is 0, and the weight of an edge between u and a vertex $v \in (\bigcup \mathcal{B}) \setminus B(u)$ is the same as the weight of an edge between any vertex in $B(u)$ and v. Since $\mathsf{distinct}(H) = H$, all the vertices in H have distinct weighted neighborhoods, and therefore every vertex u fits at most one bag. If there exists an undecided vertex u that fits no bag, we branch and decide on a bag for u, put u in this bag, and perform the necessary changes of weights of edges between u and vertices in $\bigcup \mathcal{B}$, updating k accordingly. Here, we have b branches, and at each branch the weights of all of the edges between u and at least one bag are changed, and therefore the parameter decreases by at least s. That is, we obtain a branching vector at least as good as $(s, \ldots, s)$, where s appears b times. Below we will obtain a worse branching vector.

After the last step, every undecided vertex u fits exactly one bag $B(u)$. If there are no two undecided vertices u and v such that if we put u in $B(u)$ and v in $B(v)$, no conflict is created (that is, the weight of the edge between u and v is the same as the weight of any edge between a vertex in $B(u)$ and a vertex in $B(v)$), we can simply return the answer YES. Indeed, in this case, since $k \geq 0$ (else recall that we would have already returned NO), we have used at most k changes to modify G to a graph G' such that $\mathsf{distinct}(G') = H$ – each undecided vertex can be put in the only bag it fits and no changes are required. Therefore, we now suppose that there are two undecided vertices u and v that do create a conflict. We apply a branching rule that is an exhaustive search consisting of the following branches:

1. In the first branch, we address the conflict by changing the weight of the edge between u and v to the weight of any edge between a vertex in $B(u)$ and a vertex in $B(v)$, decrease k by 1, and then put u in $B(u)$ and v in $B(v)$.
2. Next, we consider $b-1$ branches that find a new bag for u. More precisely, in each of these branches, we put u in a different bag B_i in $\mathcal{B} \setminus \{B(u)\}$, update the weights of the edges between u and other vertices in $\bigcup \mathcal{B}$ accordingly (that is, if the weight of an edge between u and a vertex in B_i is not 0, we update it to 0, and if the weight of an edge between u and a vertex in a different bag B_j is not the same as the weight of the edge $\{v_i, v_j\}$ in H, we update it to this weight). Moreover, in each of these branches, we decrease k by the number of the changes that were made. Observe that since u fits only $B(u)$, in each of these branches k is decreased by at least s.
3. Finally, we consider $b-1$ branches that find a new bag or v. These branches are symmetric to those considered in the previous item.

The branching vector we obtain is at least as good as $(1, s, \ldots, s)$, where s appears $2(b-1)$ times. By Proposition 7, the branching number is bounded by $1 + \dfrac{\log_2 s}{s}$.

Handling the closed bags: The closed bags do not guarantee branching vectors as good as those given previously. For example, when we change all of the weights of the edges between some undecided vertex u and the vertices of a closed bag, we are changing less than s values, and therefore the drop in k is smaller than s. However, the closed bags do not really pose a problem due to the following arguments, which rely on the fact that no undecided vertex should be put into them. We will show that each closed bag can be associated with some open bag, and that changes related to closed bags can be done after all of the changes related to open bags have been considered and there are no more undecided vertices.

Let U be the set of vertices of H' corresponding to the open bags. Note that $H'[U]$ may not remain a graph satisfying $\mathsf{distinct}(H'[U]) = H'[U]$. In that case we group together bags which are equivalent in $H'[U]$ and call each group a superbag, where two bags B_i an B_j are equivalent if the weight of each edge between them is 0, and for every bag B_ℓ, the weight of an edge between a vertex in B_i and a vertex in B_ℓ is the same as the weight any edge between a vertex in B_j and a vertex in B_ℓ. So each bag in $H'[U]$ is either a "normal" bag or a superbag. Observe that each bag in $H'[U]$ has at least s vertices in it. Considering the graph $H[U]$ with the superbags, we perform exactly the same branching rules as above, where we assume that all the bags are open. We have fewer branches and the branching vectors do not get worse. After branching, at the point where we have previously returned YES, we have that the each undecided vertex fits one of the bags in $H'[U]$ and the weights of the edges between them are fixed without conflicts. Now, while actually adding a vertex u to a bag we also decide on the bag within a superbag S that will host u. Adding u to a bag does not change the weights of edges within the union of open bags and set of undecided vertices. Therefore we can take these decisions independently for each u, adding u to any bag in S that causes the minimum number of changes of weights of edges between u and vertices in the closed bags, and updating k accordingly.

Time complexity: Recall that the preprocessing phase is performed in the desired time. Thus, it remains to analyze the time necessary to perform the calculations following this phase. The worst branching number we obtained was bounded by $1 + \frac{\log_2 s}{s}$, assuming that s is significantly larger than r and p. Therefore, since $s = \lfloor\sqrt{k}\rfloor$, we obtain that the running time of our algorithm is bounded by $\mathcal{O}^*(2^{f(r,p)} \cdot (1 + \frac{\log_2 s}{s})^k) = \mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$ for some function f. ◀

E Bounded Search Trees

Informally, a *bounded search tree* or *branching* is used to represent the execution of an algorithm which solves a problem based on the solution of sub-problems. It can be represented as a tree and the algorithm can be imagined to solve the sub-problems one at a time by traversing this tree. The correctness of a branching algorithm can be justified by arguing that in the case of a YES-instance some sequence of decisions captured by the algorithm leads to a feasible solution. The running time of the algorithm is given by the size of the branching tree. For a parameterized instance, if the size of the branching tree is bounded by a function of the parameter and and each step of the algorithm takes polynomial time then such a branching algorithm leads to an FPT algorithm.

One method to bound the size of a branching tree employs the notion of *branching vectors*. To each node of the tree we associate a value using a function which depends on the instance to be solved at that node. This function, usually referred to as a *measure function*, is set up in such a way that it takes a smaller value for a sub-problem. It should also satisfy the property that it is a bounded function. Now the size of the tree can be upper-bounded by looking at the drop in value of the measure function at each branch of a node. This drop is represented as a tuple of numbers and is referred to as a *branching vector*. Formally, suppose that the algorithm executes a rule which has ℓ branches (each corresponding to a recursive call), such that in the i^{th} branch, the current value of the measure decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is the branching vector of this rule. We say that α is the *branching number* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \ldots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the measure, and the algorithm returns a result when (or before) it becomes negative, the running time is bounded by $\mathcal{O}^*(\alpha^r)$. For more details we refer the reader to [3].