Deterministic Network Coding by Matrix Completion

by

Nicholas James Alexander Harvey
Bachelor of Mathematics, University of Waterloo, 2000

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005

Author . . . .
Department of Electrical Engineering and Computer Science
May 06, 2005

Certified by . . . .
David R. Karger
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Deterministic Network Coding by Matrix Completion

by

Nicholas James Alexander Harvey

Bachelor of Mathematics, University of Waterloo, 2000

Submitted to the Department of Electrical Engineering and Computer Science
on May 06, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Network coding is a new field of research that addresses problems of transmitting data through networks. Multicast problems are an important class of network coding problems where there is a single sender and all data must be transmitted to a set of receivers. In this thesis, we present a new deterministic algorithm to construct solutions for multicast problems that transmit data at the maximum possible rate. Our algorithm easily generalizes to several variants of multicast problems.

Our approach is based on a new algorithm for *maximum-rank completion of mixed matrices*—taking a matrix whose entries are a mixture of numeric values and symbolic variables, and assigning values to the variables so as to maximize the resulting matrix rank. Our algorithm is faster than existing deterministic algorithms and can operate over smaller fields. This algorithm is extended to handle collections of matrices that can share variables. Over sufficiently large fields, the algorithm can compute a completion that simultaneously maximizes the rank of all matrices in the collection.

Our simultaneous matrix completion algorithm requires working over a field whose size exceeds the number of matrices in the collection. We show that this algorithm is best-possible, in the sense that no efficient algorithm can operate over a smaller field unless P=NP.

Thesis Supervisor: David R. Karger
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

The material of this thesis is primarily joint work with David Karger and Kazuo Murota. A preliminary version of this work was presented at the SODA '05 conference [17]. I would like to thank David Karger for introducing me to the field of network coding, and posing the problem which is addressed in this thesis. This thesis would not have been possible without his advice, support, and endless stream of research ideas. I am also very grateful to Kazuo Murota for corresponding with me about the field of mixed matrices. His mathematical virtuosity was crucial for many of the ideas in Chapter 3, particularly Section 3.6.

I would also like to thank my friends here at MIT. My collaborations with Kevin Zatloukal, April Lehman and Bobby Kleinberg were thoroughly enjoyable and I learned much from them. I am deeply indebted to April for painstakingly reading a draft of this thesis and providing substantial amounts of helpful feedback. A valuable conversation with Sergey Yekhanin and David Woodruff catalyzed the results in Section 4.2.

Finally, I am grateful to my family for their love and support. My wife Junko has been an unwavering source of encouragement, emotional support and happiness throughout the years. None of this would have been possible without her.

This thesis is dedicated to the memory of my mother, Gail Harvey.

Contents

1 Introduction **9**

1.1 Network Coding 10

1.2 Matrix Completion 12

1.3 Our Results 14

2 Multicast Network Coding **17**

2.1 Preliminary Definitions 17

2.2 Related Work 18

2.3 Koetter and Médard's Algebraic Framework 22

2.4 The Expanded Transfer Matrix 24

2.5 An Algorithm for Multicast Problems 25

2.6 Variants of Multicast Problems 27

3 Matrix Completion **31**

3.1 Related Work 31

3.2 Mixed Matrices 33

3.3 Computing the Rank of an LM-matrix 36

3.4 Single Matrix Completion 39

3.4.1 A Simple Algorithm 39

3.4.2 A Better Algorithm 40

3.4.3 Efficiency of the Better Algorithm 45

3.5 Simultaneous Matrix Completion 49

3.5.1 Column Compatibility 52

3.6 Single Matrix Completion, Revisited 52

4 Hardness of Matrix Completion **55**
4.1 Simultaneous Completion when $q < d$ 55
4.2 Simultaneous Completion when $q = d$ 60
4.3 Deciding Non-Singularity 63

A Preliminaries on Linear Algebra **67**

B Preliminaries on Matroids **73**
B.1 Definitions 73
B.2 Bipartite Matchings and Transversal Matroids 76
B.3 Algorithms 77

Bibliography **80**

Chapter 1

Introduction

Sending data through a network is a task that pervades much of our modern lives. The question of how to send data efficiently remains a central question in several fields of research, from information theory to computer networking. Despite the considerable amount of research that has focused on this question, many aspects of it remain poorly understood.

Even choosing an appropriate mathematical model for data transmission problems is difficult. For a computer scientist, a natural approach would be to model the problem using network flows or some other combinatorial packing problem. However, such models are often completely inappropriate.

> *The theory of information flow in networks does not have the same simple answers as the theory of flow of water in pipes.*
>
> Cover and Thomas, *Elements of Information Theory* [7]

Why is information different from water? One key difference is that data can be manipulated in ways that water and other physical commodities cannot. For example, data is often redundant and can therefore be compressed. Even incompressible data streams can be manipulated in interesting ways, such as coding several streams together. For example, analog signals can be combined using frequency-division multiplexing, and digital signals can be combined by algebraic operations, such as computing their XOR (exclusive-or). We will show shortly that coding data streams together can offer significant benefits in certain cases.

The area of information theory that addresses data communication in networks is called ***network information theory***. The problems in this area often involve extremely complex issues, such as dealing with noise in the network, interference between signals, and correlation between data sources. As a result, many problems in the area that superficially appear quite simple have remained unsolved for decades.

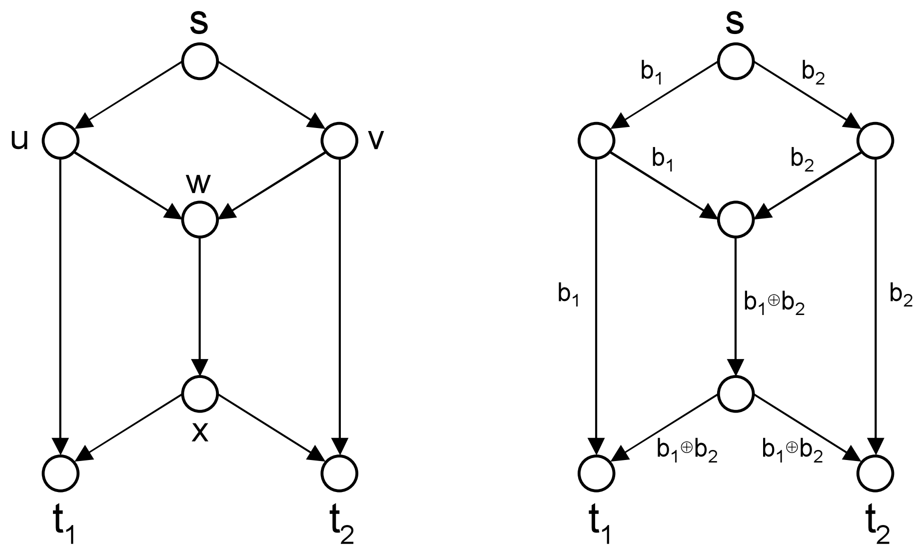
The seminal paper of Ahlswede et al. [1] suggested a way around the difficulties of network information theory. Their work begins with the simple observation that noise and interference can be avoided in practice through the use of appropriate protocols at the physical layer. They then turn to the question of whether coding data streams together still provides any benefit, even in the absence of noise. The surprising result is that coding does indeed provide a benefit. This observation generated considerable interest and lead to a new area of research called ***network coding***, which we describe in the following section.

This thesis focuses on a class of network coding problems called ***multicast problems***, where a single sender must transmit all its data to a set of receivers. A network coding solution to multicast problems can be found by completing the entries in a set of partially specified matrices so that the resulting matrices have full rank. This approach was previously used to obtain a randomized algorithm for solving multicast problems [19]. We show that this matrix problem can also be solved deterministically, thereby implying a new deterministic algorithm for solving multicast problems. Our algorithm has the restriction that it must operate over a sufficiently large field. We show that this restriction is optimal — no efficient algorithm can operate over a smaller field unless P=NP.

1.1 Network Coding

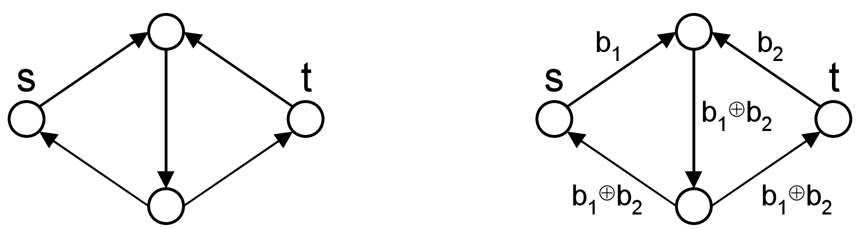
The paper of Ahlswede et al. contained a simple but compelling example which shows the power of network coding. This example, shown in Figure 1-1 (a), has come to be called the ***Butterfly Graph***[1]. Each edge represents a communication link with one-bit capacity. The source s wants to send two bits b_1 and b_2 simultaneously to both of the sinks t_1 and t_2. Without network coding, one can show that this is impossible. (We will forgo this argument and present an even simpler example in Figure 1-2.) In the network coding model, the internal nodes of the network

[1]This is unrelated to the butterfly network that is studied in the field of parallel computing [28].



(a) (b)

Figure 1-1: (a) The Butterfly Graph. (b) The network coding solution.



(a) (b)

Figure 1-2: (a) The Wheatstone Bridge. (b) The network coding solution.

are allowed to perform arithmetic operations on the bits, and thus a solution is possible. Figure 1-1 (b) illustrates this solution, where $\oplus$ denote the binary XOR operation.

We present a simple variant of the Butterfly Graph in Figure 1-2 (a). This graph is often called the Wheatstone bridge, due to its similarity with the famous electrical network [48]. In this example, we wish to send the bit b_1 from node s to node t, and simultaneously send the bit b_2 from node t to node s. Without network coding, this is clearly impossible since there does not exist a disjoint pair of s-t and t-s paths. In the network coding model, Figure 1-2 (b) shows that it is possible to send the XOR $b_1 \oplus b_2$ to both nodes simultaneously. Node s already knows the value of

b_1 so it can recover the value of b_2 by computing $b_1 \oplus (b_1 \oplus b_2)$. Node t can similarly recover the value of b_1.

These examples suggest that the network coding model is an intriguing framework with many surprising implications for data transmission problems. Much of the existing network coding literature has focused on special classes of network coding problems, particularly ***multicast problems*** on directed acyclic graphs (DAGs). A multicast problem is an instance with a single commodity, a single source node, and an arbitrary set of sink nodes. The objective is to transmit all the information available at the source to all of the sinks. We have already shown an example of a multicast problem, namely the Butterfly Graph.

Given an instance of a multicast problem on a DAG, a natural question is: How can one transmit data from the sources to the sinks at the maximum possible rate? This is one of the central questions addressed by this thesis. Koetter and Médard [23] showed that multicast problems can be recast in an algebraic framework involving matrices of indeterminates[2]. Using their framework, we can recast the multicast network coding question as the problem of finding a "matrix completion". We elaborate on the latter problem in the next section.

1.2 Matrix Completion

Suppose one is given a matrix M whose entries are a mixture of numbers and distinct indeterminates. Such a matrix is called a ***mixed matrix***. Sometimes it is inconvenient to deal with mixed matrices, so one might want to plug in particular values for the indeterminates. Such an assignment of values to the indeterminates is called a ***completion***. One might imagine looking for completions that satisfy a variety of properties, such as maximum rank, positive definiteness, a desired spectrum, etc. A comprehensive survey of these completion problems is given by Laurent [25].

Example 1.1. Consider the following mixed matrix.

$$M = \begin{pmatrix} 1 & x \\ y & 1 \end{pmatrix}$$

[2] We use the term "indeterminate" to mean an independent variable.

Assigning the values $x = 1$ and $y = 1$ gives a completion of the matrix. However, this is not a maximum rank completion since the resulting matrix has rank 1. The assignment $x = 0$ and $y = 0$ is a maximum rank completion since the resulting matrix is the identity. ■

This thesis considers only the problem of finding completions with maximum rank. For brevity we will henceforth use the term "completion" to mean one with maximum rank. The problem of finding a completion has been considered for many years and has a surprising number of important applications. For example, completions are a key ingredient in parallel algorithms for finding maximum matchings [34], sequential algorithms for finding maximum matchings [33], and dynamic algorithms for transitive closure [42].

How can one find a completion? Lovász [30] pointed out that simply choosing values at random from a sufficiently large field gives a completion with high probability. This algorithm takes only $O(n^2)$ time to find the completion and $O(n^3)$ time if one must verify the rank of the resulting matrix. It can operate over a field of size cn for any $c > 1$.

Lovász's work left open the question of finding an efficient deterministic algorithm for constructing matrix completions. This question was solved some twenty years later by Geelen [15]. He devised an elegant algorithm that blends ideas from combinatorics, matroid theory and linear algebra, but unfortunately requires $O(n^9)$ time. Some later improvements [16, 3] reduced the running time to $O(n^4)$. Geelen's algorithm also requires working over a field of size at least n.

The connection to network coding involves a generalization of the maximum-rank matrix completion problem. Suppose one is given a set of mixed matrices, each of which contains a mixture of numbers and indeterminates. Each particular indeterminate can only appear once per matrix but may appear in several matrices. The objective is to find values for these indeterminates that simultaneously maximize the rank of all matrices. We call this a ***simultaneous matrix completion problem***. Lovász's randomized approach trivially extends to handle this problem by simply working over a larger field. This approach was used by Ho et al. [19] to obtain a randomized algorithm for multicast network coding problems. Geelen's deterministic algorithm can also be extended to compute simultaneous matrix completions, with a corresponding increase in field size and runtime.

1.3 Our Results

This thesis addresses the separate, but related problems of multicast network coding and matrix completion. The former problem is addressed in Chapter 2 and the latter problem is discussed in Chapters 3 and 4. We discuss the background and previous work for each problem in the corresponding chapters.

Chapter 2 begins with a more detailed discussion of multicast network coding. We then explain how our theorems on matrix completion lead to a new deterministic algorithm for constructing coding functions that allow communication at the maximum rate. This algorithm requires time $O(dm^3 \log m)$, where m is the number of edges in the graph and d is the number of sinks. This algorithm also shows that every multicast problem has a solution over any field of size q where $q > d$. We conclude this chapter by describing how this algorithm can be applied to several variants of the multicast problem.

Chapter 3 contains our algorithms for matrix completion problems. We begin with a discussion of existing work. Then we describe mathematical tools that we will need from the theory of mixed matrices. We then use these tools to obtain a new deterministic algorithm for computing a matrix completion. This algorithm yields the following theorem.

Theorem 1.2. *Given a mixed matrix of size $n \times n$ over any field, a max-rank completion can be deterministically computed in time $O(n^3 \log n)$. Using fast matrix multiplication, the time required is only $O(n^{2.77})$.*

This theorem shows that our algorithm can operate over smaller fields than all existing algorithms (randomized or deterministic) and is asymptotically faster than existing deterministic algorithms. We then generalize this algorithm to handle simultaneous matrix completion problems. This yields the following theorem.

Theorem 1.3. *Let $\mathcal{A}$ be a set of d mixed matrices of size $n \times n$. Let k be the number of indeterminates in these mixed matrices. A simultaneous completion for $\mathcal{A}$ necessarily exists if the mixed matrices in $\mathcal{A}$ are over a field of size strictly greater than d. Furthermore, such a completion can be deterministically computed in time*

$$O(\, d\,(n^3 \log n + k\, n^2)\,).$$

The runtime of the simultaneous matrix completion algorithm can be improved somewhat for sets of matrices satisfying a certain technical condition. Suppose that for any pair of indeterminates, whenever they appear in the same matrix, they necessarily appear in the same column. Such matrices are called ***column-compatible***. We show that, for a set of column-compatible matrices, a simultaneous completion can be found in time

$$O(\,d\,(n^3 \log n + k\,n)).$$

Theorem 1.3 may seem somewhat restrictive since it requires that the field size exceed the number of matrices. Surprisingly, this restriction cannot be removed. We show in Chapter 4 that the hardness of finding a simultaneous completion crosses a sharp threshold when the field size equals the number of matrices.

Theorem 1.4. *Let $\mathcal{A}$ be a set of d mixed matrices over a field of size q.*

- *If $q > d$ then a simultaneous completion necessarily exists and one can be computed in polynomial time.*
- *If $q \leq d$ then a simultaneous completion may not exist. Furthermore, deciding whether a simultaneous completion exists is NP-complete.*

The first claim of this theorem is simply a restatement of Theorem 1.3. The second claim shows that the field size required by Theorem 1.3 is optimal, under the assumption that P≠NP.

Chapter 2

Multicast Network Coding

We begin this chapter by providing some preliminary definitions and then surveying the existing work on multicast network coding problems. Next we give the formal definitions that are needed to present our algorithm. Then we describe our polynomial time algorithm for multicast problems. We finish by showing how the algorithm can be applied to variants of the problem.

2.1 Preliminary Definitions

Definition 2.1. *A **multicast problem instance** is a directed acyclic graph $G = (V, E)$ with a distinguished vertex $s \in V$ and a distinguished set $T = \{t_1, \ldots, t_d\} \subset V$ where $s \notin T$. The vertex s is called the **source** and the vertices in T are called **sinks**.*

The interpretation of a multicast problem is that the source s wishes to transmit information to sinks through a communication network represented by the graph G. We interpret the edges as communication channels with equal (unit) capacity. We assume that the source has a large amount of information that it wishes to transmit to the sinks. The information is divided into messages drawn from some alphabet, and we assume that each edge can transmit one symbol from that alphabet in each time step. We also assume that communication across edges is instantaneous, i.e., the edges have zero-delay.

We must also explain how communication schemes in the network are modeled. We give only informal definitions at the present time, and defer the formal definitions to Section 2.3. The communication scheme is modeled by functions at each node in the network that specify how the information received by that node is to be coded together and sent on the node's outgoing edges. A set of coding functions that specify the information sent on each edge of the graph is called a ***network code***. If a network code allows each sink to decode all messages sent by the source then it is called a ***network coding solution***.

The following definition gives a simple class of network codes. As explained in the following section, this class is particularly important for multicast problems.

Definition 2.2. *A network code is called* ***linear*** *if the alphabet is treated as a finite field and the message sent on every edge is a linear combination of the source messages.*

In this chapter, we use the following notation. Let $m := |E|$ be the number of edges in the graph, $d := |T|$ the number of sinks and r the capacity of the minimum cut separating the source from any sink. Let $\Gamma^{\text{in}}(v)$ and $\Gamma^{\text{out}}(v)$ denote the inbound and outbound edges at vertex v. For an edge $e \in E$, let $\Gamma^{\text{in}}(e)$ denote the edges inbound to the tail of e.

2.2 Related Work

The network coding model was first introduced in the seminar paper of Ahlswede et al. [1]. The intriguing examples presented in their paper, such as the Butterfly Graph (Figure 1-1), generated much interest in network coding. Curiously, the Butterfly Graph was already known to Hopcroft in the mid 1980s [20]. This earlier discovery arose from different motivations, and ultimately lead to the development of superconcentrators [38].

A wide variety of communication problems can be considered within the network coding model. For example, an instance of a general network coding problem can involve an arbitrary graph and an arbitrary number of commodities, each with arbitrary sets of sources and sinks. Such general problems are quite difficult to analyze and remain poorly understood. Some of the difficulties are foundational: even defining network coding in cyclic graphs is non-trivial since one must ensure that the coding functions do not satisfy any impossible cyclic relationships. For a discussion of these issues see, for example, Harvey et al. [18].

Due to the difficulty of general network coding problems, much work has focused on the multicast problem, which was also introduced by Ahlswede et al.

They proved the following important theorem.

Theorem 2.3 (Ahlswede et al. [1]). *The maximum rate of simultaneous communication from the source to every sink equals the capacity of the minimum cut separating the source from some sink.*

The conclusion of this theorem is perhaps quite intuitive: it sounds very similar to the well-known max-flow min-cut theorem, due to Ford and Fulkerson [11] and Elias, Feinstein and Shannon [9]. However, there are two surprising aspects to Theorem 2.3. First, the conclusion of the theorem is false when coding is not permitted, as the Butterfly Graph illustrates. Second, network coding solutions can actually achieve the combinatorial min-cut bound, whereas combinatorial path-packing solutions cannot. Theorem 2.3 may be informally summarized as saying "max-flow min-cut is false for multicast problems, unless network coding is used".

Since the min-cut separating a pair of vertices can be computed in polynomial time, Theorem 2.3 implies that the maximum number of messages that can be multicast can also be computed in polynomial time. Although this theorem fully characterizes the maximum communication rate, it does not tell us how to find a network coding solution.

There is a potential obstacle to finding a network coding solution efficiently: the description of the solution might be very large. First note that the alphabets used by a network coding solution can be arbitrarily large, so there are in fact infinitely many solutions. One might expect that it is sufficient to consider only exponential-sized alphabets, but surprisingly this is not the case. Lehman and Lehman [27] observed that an alphabet of doubly exponential size is necessary to achieve the maximum rate in certain non-multicast network coding instances.

Even if we restrict ourselves to very small alphabets, we can still run into difficulties. Consider a graph with n vertices and suppose we restrict ourselves to alphabets of size 2. Suppose that there is a vertex v with $\Theta(n)$ inbound edges. The coding function for any outbound edge of v is a map from a set of size $N = 2^{\Theta(n)}$ to a set of size 2. There are 2^N such maps and hence there must be some map whose representation requires at least N bits, which is exponential in the size of the instance. Thus it is not even clear that we can write down a network coding solution in polynomial time.

Fortunately, Li et al. [29] showed that we can restrict our attention to linear network codes. As will be made clear in Section 2.3, the space required to represent a linear network code is only polynomial in the size of the instance and the logarithm of the alphabet size. Furthermore, we will see that it is sufficient to work with an alphabet whose size is polynomial in the size of the instance.

Theorem 2.4 (Li, Yeung and Cai [29]). *Every multicast problem has a linear network coding solution over some alphabet.*

An elegant algebraic framework for dealing with linear network codes was developed by Koetter and Médard [23]. We now give an informal discussion of their framework and defer a detailed discussion to the following section. Notice that when a linear network coding solution is used, the messages arriving at the sinks are just linear functions of the source nodes' messages. Koetter and Médard show how these linear combinations at each sink can be explicitly described by a so-called ***transfer matrix*** whose entries are determined by the linear functions selected at each node. Intuitively, if the linear combinations received by each sink have full rank then the sinks can invert the linear combinations and recover the original source messages. The problem of selecting a network code thus reduces to choosing appropriate entries for each transfer matrix. The challenge is to select entries for the transfer matrices so that this full rank condition holds at each sink simultaneously. This problem is made harder by the requirement that the entries in different matrices must "match".

The preceding discussion suggests that finding a network coding solution to a multicast problem is closely related to the matrix completion problems described in Section 1.2. The subsequent sections will elucidate this connection. Ho et al. [19] used this reduction to matrix completion to develop an efficient randomized algorithm for multicast problems. The basic idea of their work is to show that choosing a random completion over a sufficiently large field gives a network coding solution with high probability.

A deterministic polynomial time algorithm for finding network coding solutions for multicast problems was concurrently developed by Sanders et al. [41] and Jaggi et al. [21]. This algorithm is also described in a joint journal publication [22]. A brief description of their algorithm is as follows. First, it computes maximum flows from the source to each sink. Next, it computes a coding scheme such that for any source-sink cut, the messages sent across the cut are sufficient to reconstruct the original source messages. Fortunately, their algorithm need not consider all exponentially many cuts; it suffices to consider the cuts encountered during a breadth-first search from the source towards the sinks.

One of the main contributions of this thesis is a new deterministic algorithm for finding network coding solutions for multicast problems. We derive this algorithm by derandomizing the randomized matrix completion approach of Ho et al. Although the algorithm of Jaggi et al. is somewhat faster than ours, the generality of our approach allows straightforward extensions to several variants of the problem. For example, we can construct one code that can be used unchanged *regardless* of which node is the multicast source. We also give a deterministic construction of network codes for the ***robust multicast problem*** [23], which seeks network coding solutions that can handle unpredictable failures of some edges. These extensions are discussed in Section 2.6.

An important benchmark for comparing network coding solutions is the size of the alphabet used. A large alphabet provides more flexibility in choice of coding functions, but causes greater cost in storing and computing messages. Thus it is desirable to work with an alphabet that is large enough to enable transmission at the maximum rate, but not much larger. Recall that d denotes the number of sinks in a multicast problem instance. For multicast problems, it is known that an alphabet of size $d+1$ is sufficient to transmit at the maximum rate [19] and that $\Omega(\sqrt{d})$ is necessary for some instances [26, 10]. The algorithm that we present also works with an alphabet of size $d+1$. Jaggi et al. [22] actually obtained a slightly tighter bound: an alphabet of size d is sufficient. However, these algorithms mentioned above all work over a finite field, and hence they must round the field size up to a prime power. Thus the distinction between alphabets of size d and $d+1$ is irrelevant unless d is a prime power.

2.3 Koetter and Médard's Algebraic Framework

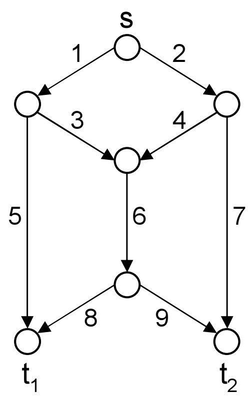
In this section we formally introduce Koetter and Médard's algebraic framework for multicast problems. We assume that the source has r messages $M_1, \ldots, M_r$ that it wishes to transmit to the sink nodes, where r is the min-cut value as defined earlier. The messages that are output by the i^{th} sink are denoted by $T(i, j)$ for $1 \leq j \leq r$. Let f_e denote the message transmitted by edge e. For any network code, the message f_e must be computable from the messages on the edges in $\Gamma^{\text{in}}(e)$. Thus for linear network codes we obtain the following system of linear equations, where the α's, β's and γ's are indeterminates.

$$
\begin{array}{lll}
\text{Source edge } e \in \Gamma^{\text{out}}(s): & f_e = & \displaystyle\sum_{1 \leq j \leq r} \alpha_{j,e} \, M_i \\
\text{Sink message } T(i,j): & T(i,j) = & \displaystyle\sum_{e' \in \Gamma^{\text{in}}(t_i)} \beta_{e',i,j} \, f_{e'} \\
\text{Edge } e \in E \setminus \Gamma^{\text{out}}(s): & f_e = & \displaystyle\sum_{e' \in \Gamma^{\text{in}}(e)} \gamma_{e',e} \, f_{e'}
\end{array}
$$

The coefficients may be collected into matrices as follows. There is an $r \times m$ matrix $A := (\alpha_{j,e})$. For $1 \leq i \leq d$, there is an $m \times r$ matrix $B_i := (\beta_{e',i,j})$. Finally, there is an $m \times m$ matrix $F := (\gamma_{e',e})$.

We will now illustrate these definitions with an example.

Example 2.5. Consider again the Butterfly Graph. Here we have numbered its edges from 1 through 9.



The corresponding matrices are as follows. An absent entry denotes a zero.

$$A = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & & & & & & & \\ \alpha_{2,1} & \alpha_{2,2} & & & & & & & \end{pmatrix}$$

$$F = \begin{pmatrix} 1 & & \gamma_{1,3} & & \gamma_{1,5} & & & & \\ & 1 & & \gamma_{2,4} & & & \gamma_{2,7} & & \\ & & 1 & & & \gamma_{3,6} & & & \\ & & & 1 & & \gamma_{4,6} & & & \\ & & & & 1 & & & & \\ & & & & & 1 & & \gamma_{6,8} & \gamma_{6,9} \\ & & & & & & 1 & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{pmatrix} \quad B_1 = \begin{pmatrix} & \\ & \\ & \\ & \\ \beta_{5,1,1} & \beta_{5,1,2} \\ & \\ & \\ \beta_{8,1,1} & \beta_{8,1,2} \\ & \end{pmatrix} \quad B_2 = \begin{pmatrix} & \\ & \\ & \\ & \\ & \\ & \\ \beta_{7,2,1} & \beta_{7,2,2} \\ & \\ \beta_{9,2,1} & \beta_{9,2,2} \end{pmatrix}$$

■

Koetter and Médard introduced the following definition.

Definition 2.6. *The matrix $M_i := A(I - F)^{-1}B_i$ is called the **transfer matrix** for the sink t_i.*

This definition assumes that $(I - F)$ is non-singular. Since the graph G is assumed to be acyclic, we may assume that the edges are ordered in topological order. Thus the matrix F is strictly upper-triangular, and hence the assumption that $(I - F)$ is non-singular is necessarily satisfied. The usefulness of transfer matrices is shown by the following theorem.

Theorem 2.7 (Koetter and Médard [23]). *A multicast problem has a network coding solution if and only if each transfer matrix M_i is formally non-singular (i.e., $\det M_i$ is not the zero polynomial). Moreover, an assignment of values to the coefficients causes all the transfer matrices to be non-singular if and only if these values constitute a network coding solution.*

Transfer matrices are somewhat unwieldy since they involve the inverse of the large symbolic matrix $(I-F)$. Ho et al. [19] defined a convenient variant of transfer matrices, which we have termed expanded transfer matrices. These are discussed in the next section.

2.4 The Expanded Transfer Matrix

Definition 2.8. *The **expanded transfer matrix** for sink t_i is*

$$N_i \;:=\; \begin{pmatrix} A & 0 \\ I-F & B_i \end{pmatrix}.$$

We remark that an expanded transfer matrix N_i is a mixed matrix in the indeterminates $\alpha_{j,e}, \beta_{e',i,j}, \gamma_{e',e}$. Some entries of N_i contain numbers, namely the 1's in the identity matrix I. An important connection between transfer matrices and expanded transfer matrices is given by the following lemma, which was observed by Ho et al. [19].

Lemma 2.9. $\det M_i = \pm \det N_i$.

We present a proof of this lemma that is substantially simpler than the original proof of Ho et al. Our proof relies only on the basic properties of determinants stated in Appendix A, such as the block determinant fact (Fact A.10). We remark that this fact cannot immediately be applied to N_i because the submatrices A and B_i are not necessarily square.

Proof. Block multiplication shows that the following equation holds.

$$\begin{pmatrix} A & 0 \\ I-F & B_i \end{pmatrix} \cdot \begin{pmatrix} (I-F)^{-1}B_i & (I-F)^{-1} \\ -I & 0 \end{pmatrix} \;=\; \begin{pmatrix} M_i & A(I-F)^{-1} \\ 0 & I \end{pmatrix}.$$

Take the determinant of both sides and use the fact that $\det(XY) = \det X \det Y$ to conclude that

$$\det \begin{pmatrix} A & 0 \\ I-F & B_i \end{pmatrix} \cdot \det \begin{pmatrix} (I-F)^{-1}B_i & (I-F)^{-1} \\ -I & 0 \end{pmatrix} = \det \begin{pmatrix} M_i & A(I-F)^{-1} \\ 0 & I \end{pmatrix}.$$

Applying the block determinant fact (Fact A.10) to the second and third matrices, we obtain

$$\det N_i \cdot \big(\pm \det -I \cdot \det (I-F)^{-1} \big) = \det M_i \cdot \det I.$$

Since F is assumed to be strictly upper-triangular, $\det(I-F) = 1$. Using the fact that $\det(I-F)^{-1} = (\det(I-F))^{-1}$, we obtain that $\det M_i = \pm \det N_i$. ■

Although this lemma is simple, it is crucial to the approach of our algorithm. We therefore indulge ourselves by presenting an alternative proof.

Proof. We perform the following computation.

$$\begin{aligned} \det N_i &= \pm \det \begin{pmatrix} 0 & A \\ B_i & I-F \end{pmatrix} \\ &= \pm \det(I-F) \cdot \det \big(-A(I-F)^{-1}B_i \big) \\ &= \pm \det M_i. \end{aligned}$$

The first equality follows because swapping columns of N_i only affects the sign of the determinant. The second equality follows from the properties of Schur complements (Fact A.11). The last equality holds since the matrix F is assumed to be strictly upper-triangular and hence $\det(I-F) = 1$. ■

2.5 An Algorithm for Multicast Problems

In this section we explain how our matrix completion results stated in Chapter 1 yield a deterministic algorithm for finding a network coding solution.

The problem of finding a network coding solution for a multicast problem reduces to the problem of finding a simultaneous completion for the set of expanded transfer matrices, as is shown by Theorem 2.7 and Lemma 2.9. Given an instance of the multicast problem, let the set of expanded transfer matrices be $\mathcal{N} := \{N_1, \ldots, N_d\}$. Let $\mathcal{X}$ be the set of all indeterminates $\alpha_{j,e}, \beta_{e',i,j}, \gamma_{e',e}$ that appear in the matrices in $\mathcal{N}$. From the definition of an expanded transfer matrix it is clear that the indeterminates in each individual matrix are distinct. The matrices in $\mathcal{N}$ do share indeterminates, though: the $\alpha_{i,e}$'s and $\gamma_{e',e}$'s appear in all of the matrices. Thus the set $\mathcal{N}$ satisfies the criteria of Theorem 1.3, and hence the matrix completion algorithm can produce a simultaneous completion in the field $\mathbb{F}_q$, for any $q \geq d+1$.

Let us now analyze the runtime of this algorithm. The dimension of each expanded transfer matrix is $n \times n$ where $n = m + r$. Since the capacity r of the minimum source-sink cut is at most the number of edges in the graph, we have $n = O(m)$. The number of indeterminates is $|\mathcal{X}| \leq mr + mdr + m^2 = O(mdr + m^2)$. Thus, Theorem 1.3 gives a runtime bound of

$$O(|\mathcal{N}|\,(n^3 \log n + |\mathcal{X}|\,n^2)) \;=\; O(m^3 d(dr + m)).$$

This runtime can be improved by taking advantage of the structure of the matrices. First, note that the matrices in $\mathcal{N}$ are very similar — only the last r columns (consisting of the B_i submatrices) differ, and the indeterminates in these last columns appear only in a single matrix. Thus the collection of matrices $\mathcal{N}$ satisfies the column-compatible condition that was informally defined in Chapter 1. (The formal definition is given in Section 3.5.1) This immediately implies that the runtime may be improved to

$$O(\,|\mathcal{N}|\,(n^3 \log n + |\mathcal{X}|\,n)) \;=\; O(m^2 d(dr + m \log m)).$$

Further improvements are possible. Since each $\beta_{t,i,e'}$ appears only in one matrix, these indeterminates do not even require simultaneous completion. Thus we may find a completion of $\mathcal{N}$ in two steps. First, we remove the $\beta_{t,i,e'}$'s from $\mathcal{X}$ and find a simultaneous completion for the remaining indeterminates. Note that we do not removing the $\beta_{t,i,e'}$'s from the matrices, only from the set $\mathcal{X}$ of indeterminates that require completion. The time required for this step is $O(dm^3 \log m)$ since we now have $|\mathcal{X}| = O(m^2)$. Next, we complete the $\beta_{t,i,e'}$'s by finding a completion of each matrix separately. The total time required for this step is $O(dm^3 \log m)$. Combining these two steps solves the multicast problem in time $O(dm^3 \log m)$.

This algorithm, as described above, requires knowing the value of r, the minimum source-sink cut. We can invoke the algorithm multiple times and use binary search to find the correct value of r, since the algorithm will fail if r is too large. This binary search approach increases the running time by a logarithmic factor. A preferable approach is to simply compute the value of r using d invocations of the Ford-Fulkerson algorithm [6]. Doing so only requires $O(dm^2)$ time, which is negligible.

To compare our algorithm with existing algorithms, we note that our algorithm uses a field size that only one larger than the field size used by Jaggi et al. [22]. (The same bound was proven by Ho et al. [19], although non-algorithmically.) The algorithm of Jaggi et al. [22] requires $O(mdr(r+d))$ time, which is markedly faster when $r = o(m)$ and $d = o(m)$. An advantage of our matrix completion approach is its generality — the matrix completion approach provides a straightforward solution to several variants of the multicast problem.

2.6 Variants of Multicast Problems

In this section we discuss two variants of the multicast problem. The generality of our simultaneous matrix completion approach allows us to solve these variants quite easily. The idea behind both of these variants is that we want nodes to be oblivious to changes (e.g., failures) occurring in the network. The benefit is that changes affect only nearby nodes, and that all other nodes behave in a very predictable manner. This could be useful in a scenario when the internal network nodes are computationally limited.

One variant of multicast problems posed by Koetter and Médard [23] is the problem of designing a ***robust network code***. Formally, define a link failure pattern to be a set $F \subseteq E$ such that every edge in F fails during the operation of the network. A failure of an edge can be modeled by having the failed edge transmit the 0 message instead of the function specified by the network code. If the min-cut between the source and each sink is still at least r after the edges in F have been removed then the multicast problem on the remaining network is still solvable. A network code that solves both the original multicast problem and the multicast problem on the remaining network is desirable since the internal network nodes could then operate identically regardless of whether the failure had occurred. More generally, if $\mathcal{F}$ is a collection of link failure patterns that preserve the min-cut between the source and each sink, we desire a network code that solves the multicast problem under any failure pattern in $\mathcal{F}$.

For example, suppose we wish to handle the failure of any two edges. There are at most $\binom{|E|}{2}$ such "failed graphs" that preserve the min-cut value. We desire a single network coding solution that works for all of these failed graphs.

For any such set $\mathcal{F}$ where $|\mathcal{F}|$ is polynomial in m, the desired network code can be computed deterministically in polynomial time as follows. For every $F \in \mathcal{F}$, define $N_{t,F}$ to be the transfer matrix for sink t where the coefficients $\alpha_{j,e}, \beta_{e',i,j}, \gamma_{e',e}$ have been set to zero if edge $e \in F$. Let $\mathcal{N} := \{ N_{t,F} : t \in T, F \in \mathcal{F} \}$ be the collection of these matrices. Since each failure pattern essentially gives a new multicast problem, a simultaneous max-rank completion for $\mathcal{N}$ yields a network coding solution that works under any failure pattern in $\mathcal{F}$. This solution requires an alphabet size of at least $d \cdot |\mathcal{F}| + 1$.

This algorithm ensures that each sink's transfer matrix is invertible under any of the failure patterns in $\mathcal{F}$. This alone is not enough for the sinks to decode the source's original messages. We must also ensure that each sink knows the actual values in its transfer matrix so that it can invert the matrix. However, the values in the transfer matrix will in general depend on which failure (if any) has actually occurred. Therefore, when a node transmits a message along an edge in the network, it must also send the coefficients of the linear combination corresponding to the transmitted message. With this scheme, each sink will always know which linear combinations it is receiving, and therefore it will know how to invert them. Naturally, we can minimize the bandwidth used by these linear combinations by sending them only when they change. The preceding discussion shows that it is not strictly necessary to model a failed edge as sending the 0 message. Any other constant message would work equally well; the crucial point is that the corresponding linear combinations must also be sent.

Our algorithm above is efficient only if $|\mathcal{F}|$ is polynomial in the size of the instance. This leads to an interesting open question. Suppose that $\mathcal{F}$ contained the set of *all* failure patterns that do not decrease the minimum source-sink cut value. Note that $|\mathcal{F}|$ can be exponential in the size of the graph. Can one find a network coding solution that works under all of these failure patterns? The approach of Ho et al. [19] implies that a randomly chosen linear network code works with high probability, assuming the alphabet size is $\Omega(d \cdot |\mathcal{F}|)$. Is it necessary for the alphabet to be exponentially large, or can one find a network coding solution that uses an alphabet of polynomial size?

For our second variant of the multicast problem, consider a directed acyclic graph G with a set of sources $S := \{s_1, s_2, \ldots\}$ and a set of sinks T. Each source s_i potentially has a set of messages $\{M_{i,1}, M_{i,2}, \ldots\}$ to transmit to the sinks in T. The objective is for these sources to *sequentially* transmit their messages (if any) to the sinks: first source s_1 transmits its values, then source s_2 transmits its values, and so on. We call this the ***anysource multicast problem***. Since the sources never transmit simultaneously, the problem is essentially an ordinary multicast problem while source s_i is transmitting. Rather than treating each multicast problem separately, we desire a single network code that is a solution to each individual multicast problem.

The anysource multicast problem can also be solved with a straightforward application of simultaneous matrix completions. Define $N_{t,i}$ to be the transfer matrix for sink t when source s_i is transmitting and let $\mathcal{N} := \{\ N_{t,i}\ :\ t \in T,\ 1 \leq i \leq |S|\ \}$ be the collection of these matrices. It is easy to see that a simultaneous max-rank completion of the matrices in $\mathcal{N}$ yields a network code that solves the anysource multicast problem over any field of size at least $d \cdot |S| + 1$. As with the robust multicast problem, each transmitted message must also contain the identity of the source that is transmitting, and the coefficients of the linear combinations that are being sent.

It is not clear that our solution to the anysource multicast problem is making the most efficient use of bandwidth. Can several of the sources to transmit to the sinks simultaneously? Unfortunately, allowing several sources to transmit messages takes us beyond the realm of multicast problems. No algorithms and very few bounds on achievable rates are known for these more general network coding problems. Our algorithm for anysource multicast problems is not provably efficient, but nothing better is currently known. Any progress on algorithms or bounds for general network coding problems would be a significant development.

Chapter 3

Matrix Completion

In this chapter we describe an efficient algorithm for computing a simultaneous matrix completion. This is based on an efficient algorithm for finding a completion for a single matrix. (Recall that "completion" implicitly means that we are maximizing the rank.) Our approach for finding a completion is to partition the indeterminates into two sets. Those in the first set are deemed "unimportant" in the sense that they can immediately be assigned the value zero. For the second set of "important" indeterminates, we will give a more careful approach for choosing their values. Deciding which indeterminates are important amounts to solving a related problem called the ***matroid matching problem***. A brief introduction to the field of matroids is given in Appendix B.

We begin this chapter by describing the previous work relating to matrix completions. Next, we give an introduction to the theory of mixed matrices and to Murota's rank computation algorithm. Building on these tools, we develop our matrix completion algorithms.

3.1 Related Work

Let's begin by considering matrices where every entry is either zero or a distinct indeterminate. These matrices are much easier to deal with than general mixed matrices. As explained in Appendix B, there are deep connections between matchings in bipartite graphs and matrices containing only zeros and indeterminates. For example, suppose one is given such a square matrix A with row-set R and column-set C and one must determine whether A is singular. Given A, construct the corresponding bipartite graph

$$G_A := (R \cup C, \{(i,j) : i \in R, j \in C, A_{ij} \neq 0\}), \tag{3.1}$$

as in Definition B.3. It is well known (Theorem B.4) that there is a perfect matching in G_A if and only if A is non-singular, i.e., its determinant is the zero polynomial. Such a matching can easily be found using standard algorithms. Moreover, given a perfect matching, it is easy to give a completion for A: the indeterminates corresponding to edges in the matching are set to 1 and all others are set to 0. It is important to note that this approach does not work if the matrix contains non-zero numbers since some terms of the determinant might accidentally cancel each other out.

The reduction between matrix completion and matchings presented above is more commonly used in the reverse direction. Given a bipartite graph, deciding whether it has a perfect matching reduces to deciding whether the corresponding matrix of indeterminates is non-singular. The latter problem can be solved by computing the determinant symbolically, but this approach is not efficient since the determinant could have exponentially many terms. Alternatively, one could test singularity by trying to find a completion; Lovász's randomized algorithm [30] does precisely this. Suppose that the given matrix has dimension $n \times n$. The basic idea is to view the determinant of the matrix as a multivariate polynomial over a field of size at least cn for some constant $c > 1$. Using the Schwartz-Zippel lemma [32], one can argue that most completions are not roots of this polynomial.

We remark that Lovász's algorithm does not actually construct a perfect matching, it only decides their *existence*. Mulmuley et al. [34] build on Lovász's work and give a parallel algorithm to construct a matching, but their algorithm is not terribly efficient in a sequential setting. A much more efficient sequential algorithm was given by Mucha and Sankowski [33].

Let's now turn our attention to matrices whose entries contain a mixture of numbers and distinct indeterminates. Such matrices are called ***mixed matrices***. Does the bipartite graph of equation (3.1) allow us to determine whether a mixed matrix is singular? Unfortunately the answer is no, because the correspondence between matrices and bipartite matchings requires algebraic independence of the matrix entries. This condition will typically not hold if some entries are numeric. Thus the problems of testing singularity or finding a completion for a mixed matrix cannot directly be solved via matchings. However, these problems are efficiently solvable. Lovász's approach immediately gives a randomized algorithm for finding a completion of a mixed matrix — the presence of numeric entries in the matrix is inconsequential. More sophisticated combinatorial tools were used in developing deterministic algorithms. Murota [35, 36] showed that rank computation, and hence testing singularity, can be solved deterministically by matroid intersection. We explain this algorithm in Section 3.3. Geelen [15] developed a deterministic algorithm for actually finding a completion using different techniques.

The main result of this chapter is a new deterministic algorithm for finding completions of mixed matrices over any field in $O(n^3 \log n)$ time. In contrast, Lovász's randomized algorithm requires $O(n^3)$ time to verify the rank of the resulting matrix and, in order to have a reasonable probability of success, operates over a field of size $\Omega(n)$. Using fast matrix multiplication techniques, these running times can be improved to $O(n^{2.77})$ and $O(n^{2.38})$ respectively. Geelen's deterministic algorithm can be implemented in $O(n^4)$ time [3] and also operates over a field of size $\Omega(n)$. Thus our algorithm is faster than existing deterministic algorithms, and operates over smaller fields than any existing polynomial-time algorithms.

Our approach is to extend Murota's framework from simply computing the rank of a mixed matrix to actually finding a completion. We then extend this algorithm to finding a simultaneous completion for a collection of mixed matrices. This algorithm will complete the proof of Theorem 1.2 and Theorem 1.3.

3.2 Mixed Matrices

In this section we give a brief introduction to the theory of mixed matrices. This theory was largely developed by Murota, and a more thorough introduction can be found in his monograph [36].

The study of mixed matrices was initiated by Murota and Iri [37] as a tool for systems analysis. One of their key observations is that systems of equations describing physical systems often contain two types of numbers: "accurate numbers", which indicate topological relationships in the object being studied (such as

Kirchhoff's electrical laws), and "inaccurate numbers", which are physical quantities whose values may be imprecise or subject to random noise (such as values of resistors). Their work treats the inaccurate numbers as algebraically independent, since any dependency between their values would surely be eliminated by their inherent inaccuracy. In order to analyze such systems of equations, Murota and Iri introduced the notion of mixed matrices. On the theoretical side, much work has gone into understanding the combinatorial and structural properties of mixed matrices. On the practical side, mixed matrices have been used to solve problems in various fields, such as electrical and chemical engineering [36].

Definition 3.1. *Let $\mathbb{F}$ and $\mathbb{K}$ be fields such that $\mathbb{F}$ is a subfield of $\mathbb{K}$. A matrix A over $\mathbb{K}$ is called a* ***mixed matrix*** *if $A = Q + T$ where Q is a matrix over $\mathbb{F}$ and T is a matrix over $\mathbb{K}$ such that the set of T's non-zero entries is algebraically independent over $\mathbb{F}$.*

For the purposes of this thesis, $\mathbb{F}$ will be a finite field of numbers. Informally, we may view $\mathbb{K}$ as the same field as $\mathbb{F}$ except we have introduced indeterminates that are able to take any value in $\mathbb{F}$. We may think of each entry of T as being either zero or a single indeterminate in $\mathbb{K}$.

It is often easier to deal with mixed matrices where the numbers and indeterminates have been pulled into separate parts of the matrix. This leads to the following restricted class of mixed matrices.

Definition 3.2. *A* ***layered mixed matrix*** *(or* ***LM-matrix****) is a mixed matrix $A = Q+T$ where the non-zero rows of Q and the non-zero rows of T are disjoint. That is, A is an LM-matrix if it can be put in the form $\left(\begin{smallmatrix} Q \\ T \end{smallmatrix}\right)$.*

The class of mixed matrices is equivalent to the class of LM-matrices in the following sense. Let $A = Q + T$ be an $n \times n$ mixed matrix. The system of equations

$$Ax \;=\; b \tag{3.2}$$

is equivalent to the system of equations

$$\begin{pmatrix} I & Q \\ Z & T' \end{pmatrix} \begin{pmatrix} w \\ x \end{pmatrix} \;=\; \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{3.3}$$

where I denotes the $n \times n$ identity matrix, w is a new auxiliary vector, Z denotes the diagonal matrix $\mathrm{diag}[z_1, \ldots, z_n]$ containing new indeterminates $z_1, \ldots, z_n$, and $T'_{i,j} = -z_i T_{i,j}$. The z_i's are only introduced so that the lower half contains no numbers. Equation (3.2) is equivalent to equation (3.3) because the latter forces that $w + Qx = b$ and (after canceling the z_i's) $w - Tx = 0$, so $(Q+T)\,x = b$. The non-zero entries of T' are strictly speaking of the form $-z_i x_a$ but for notational simplicity we will assume that they are of the form x_a. This notational substitution does not affect algebraic independence of the entries.

Definition 3.3. *If A is a mixed matrix, let $\tilde{A}$ be the transformation of A used in equation* (3.3)*. We refer to $\tilde{A}$ as the **separated matrix** associated with A. The class of separated matrices is a restricted class of LM-matrices.*

We will henceforth use the notation $\left(\begin{smallmatrix} U \\ L \end{smallmatrix}\right)$ to denote an LM-matrix. We refer to U as the ***upper submatrix*** and L as the ***lower submatrix***. A separated matrix is denoted $\left(\begin{smallmatrix} I & Q \\ Z & T \end{smallmatrix}\right)$, where $\left(\begin{smallmatrix} I & Q \end{smallmatrix}\right)$ is the upper submatrix and $\left(\begin{smallmatrix} Z & T \end{smallmatrix}\right)$ is the lower submatrix. Since we view Q and T both as individual matrices and as submatrices of $\tilde{A}$, discussing their row indices and column indices can be a bit confusing. Henceforth, we will say that the row indices of Q are $\{1, \ldots, n\}$ and the row indices of T are $\{n+1, \ldots, 2n\}$. The column indices of both Q and T are $\{n+1, \ldots, 2n\}$. Thus when we say that $A = Q + T$, we actually have $A_{i,j} = Q_{i,n+j} + T_{n+i,n+j}$ and we are implicitly translating the row and column indices.

Since we showed that the systems of equations (3.2) and (3.3) are equivalent, it follows that the nullspaces of A and $\tilde{A}$ have the same dimension. Hence, the matrices A and $\tilde{A}$ satisfy the relation $\mathrm{rank}\,\tilde{A} = \mathrm{rank}\,A + n$. Thus, to compute the rank of a mixed matrix, it suffices to compute the rank of its separated matrix. To solve the latter problem, we will use the following theorem. It is phrased in terms of the larger class of LM-matrices. As defined in Appendix A, let $A[X,Y]$ denote the submatrix of A with row-set X and column-set Y.

Theorem 3.4 (Murota [36]). *Let $A = \left(\begin{smallmatrix} U \\ L \end{smallmatrix}\right)$ be an LM-matrix. Let C be the set of columns, R_U be the rows of the upper submatrix and R_L be the rows of the lower submatrix. Then,*

$$\mathrm{rank}\,A \;=\; \max_{J \subseteq C}\; \mathrm{rank}\,U[R_U, J] + \mathrm{rank}\,L[R_L, C \setminus J]. \tag{3.4}$$

Proof. We prove the result for the case that A has full rank. The more general statement follows by focusing on submatrices of A. The idea is to apply the Laplace expansion (Fact A.9) with $I = R_L$ to $\det A$. We obtain that

$$\det A \;=\; \sum_{\substack{J \subset C \\ |J|=n}} \pm \det L[R_L, J] \cdot \det U[R_U, \overline{J}]. \tag{3.5}$$

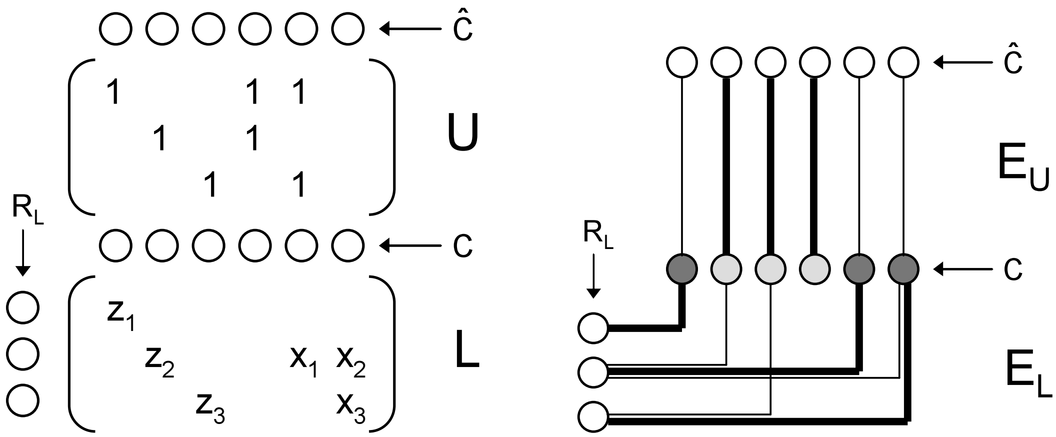
Each term of $\det L[R_L, J]$ consists of a sum of monomials that are products of variables in $\mathcal{X}$. Since each indeterminate appears only once in A, each monomial in this expansion of $\det A$ is unique. Thus there is no cancellation amongst terms of $\det A$. It follows that A has full rank whenever a pair of submatrices $L[R_L, J]$ and $U[R_U, \overline{J}]$ do. ■

The maximization performed in equation (3.4) is non-trivial, but it can be solved using matroid-theoretic algorithms. The matrix U is treated as a linear matroid $\mathbf{M}_U$ over the field $\mathbb{F}$. The matrix L is treated as the transversal matroid $\mathbf{M}_L$ for the bipartite graph corresponding to L (see equation (3.1) and Appendix B). With this formulation in terms of matroids, equation (3.4) seeks disjoint sets $J_U, J_L \subseteq C$ such that J_U is an independent set for $\mathbf{M}_U$, J_L is an independent set for $\mathbf{M}_L$, and $|J_U| + |J_L|$ is maximum. This is precisely the matroid union problem (Section B.3). Thus, the rank of a separated matrix can be computed in polynomial time using a standard algorithm for matroid union. The next section presents a refinement of this approach due to Murota [36].

3.3 Computing the Rank of an LM-matrix

We have shown that computing the rank of an LM-matrix reduces to the matroid union problem for $\mathbf{M}_U$ and $\mathbf{M}_L$. This implies an algorithm for computing the rank for any mixed matrix because every mixed matrix A is related to a separated matrix $\tilde{A}$, which is a particular type of LM-matrix.

The matroid union problem resulting from the reduction relates to matchings in two distinct ways. First, the matroid union problem can be reformulated as a matroid matching problem (see Section B.3). Second, since the matrix L contains only indeterminates and zeros, the independence oracle for $\mathbf{M}_L$ operates by computing a matching in the bipartite graph corresponding to matrix L. In this section, we will combine these two matching problems and thereby obtain a more streamlined algorithm for computing the rank of a separated matrix.



(a) (b)

Figure 3-1: (a) A separated matrix $A = \left(\begin{smallmatrix} I & Q \\ Z & T \end{smallmatrix}\right) = \left(\begin{smallmatrix} U \\ L \end{smallmatrix}\right)$ and the vertices of the corresponding matroid matching problem G. (b) The bipartite graph G for the matroid matching problem. Note that the edges E_L correspond to the non-zero entries of the lower submatrix $\left(\begin{smallmatrix} Z & T \end{smallmatrix}\right)$. The bold edges are an independent matching because columns 2, 3, and 4 of the upper submatrix are linearly independent. The light-grey vertices denote the set of columns J_U and the dark-grey vertices denote the set of columns J_L.

Given an LM-matrix $A = \left(\begin{smallmatrix} U \\ L \end{smallmatrix}\right)$, we now describe the reduction to the matroid matching problem. An example of this reduction is shown in Figure 3-1. As before, let C be the columns of A, and let R_U and R_L be the rows of U and L. Define $\hat{C}$ to be a copy of the set C where each $c \in C$ is represented as an element $\hat{c} \in \hat{C}$. The matroid matching problem instance is based on the bipartite graph $G := (V^+ \cup V^-, E_U \cup E_L)$, where

$$\begin{aligned} V^+ &:= R_L \cup \hat{C}, \\ V^- &:= C, \\ E_U &:= \{ (\hat{c}, c) : c \in C \}, \\ E_L &:= \{ (r, c) : r \in R_L, c \in C, L_{r,c} \neq 0 \}. \end{aligned}$$

The edge-set E_U simply connects corresponding columns in C and $\hat{C}$ and the edge-set E_L is precisely the bipartite matching problem associated with the matrix L. Our goal is to find a maximum cardinality matching M in G such that the columns in C covered by $M \cap E_U$ are independent in U and the columns covered by $M \cap E_L$ are independent in L. Since the matrix L contains only zeros and indeterminates, a set of columns is independent in L precisely when the vertices corresponding to those columns are matchable in the graph $G_L = (R_L \cup C, E_L)$ (see Section B.2). The columns covered by $M \cap E_L$ are clearly matchable (since they are covered by M) and hence independent in L.

To ensure that the columns covered by $M \cap E_U$ are independent in U, the edges E_U are of no use at all. However, the matroid matching problem allows us to enforce conditions on the matching M by defining matroids on the vertices of the graph. We can enforce our desired condition by defining a matroid on the vertex set $\hat{C}$. Let $\mathbf{M}_U$ be the linear matroid corresponding to the matrix U on the ground set $\hat{C}$. No additional conditions are necessary for vertex sets R_L or C, so we define $\mathbf{M}_{R_L}$ and $\mathbf{M}_C$ to be free matroids on these ground sets. To complete the definition of our matroid matching instance, we must assemble these matroids into a matroid $\mathbf{M}^+$ defined on V^+ and a matroid $\mathbf{M}^-$ defined on C. We set

$$\mathbf{M}^+ := \mathbf{M}_{R_L} \oplus \mathbf{M}_U \qquad \text{and} \qquad \mathbf{M}^- := \mathbf{M}_C.$$

Stated simply, a matching M in the graph G must only meet the requirement that the subset of $\hat{C}$ covered by M is an independent set of columns in $\mathbf{M}_U$.

Suppose we have found a maximum independent matching M in our instance. Let J_U be the set of vertices in C that are covered by $M \cap E_U$ and let J_L be the vertices in C that are covered by $M \cap E_L$. As explained above, J_U is a set of independent columns in U and J_L is a set of independent columns in L. Since J_U and J_L are disjoint, we see by equation (3.4) that $\operatorname{rank} A \geq |M|$. In fact, Murota [36] proves that $\operatorname{rank} A = |M|$ will hold. Thus to compute the rank of a separated matrix A, one can simply find a maximum independent matching in the matroid matching instance described above, and then output the cardinality of the matching. Pseudocode for this algorithm is given in Algorithm 3.1. The runtime is dominated by Step 2, which requires $O(n^3 \log n)$ time as explained in Section B.3.

Algorithm 3.1: Algorithm for computing the rank of a mixed matrix A.

Step 1: Construct the separated matrix $\tilde{A}$ and the matroid matching problem G.
Step 2: Compute a maximum independent matching M in G.
Step 3: Return $|M|$.

3.4 Single Matrix Completion

The previous section discussed how to compute the rank of a mixed matrix. We now use that framework to give an efficient algorithm for finding a max-rank completion of a single mixed matrix. To simplify the presentation, we assume throughout that the matrices given as input to the algorithm are square and have full rank. This assumption can always be satisfied by finding an appropriate submatrix of the input. We begin by discussing a simple, polynomial-time algorithm.

3.4.1 A Simple Algorithm

Let A be a mixed matrix of size $n \times n$ with full rank. We first show that A necessarily has a completion, regardless of the field from which the entries of A are drawn. Let $\mathcal{X} := \{x_1, x_2, \ldots\}$ be the set of indeterminates appearing in matrix A. Then $\det A$ is a multivariate polynomial in the variables in $\mathcal{X}$. It has total degree n but the maximum exponent of each variable is 1 since each indeterminate appears in exactly one entry of A.

The Schwartz-Zippel lemma [32] shows that if we work over a field of size $\Omega(n)$ then a constant fraction of all values for $\mathcal{X}$ are not roots of the determinant. This statement is stronger than we require: we are content to work over fields where just a single point is a non-root. The following simple lemma shows that any field will suffice.

Lemma 3.5. *Let $P(x_1, \ldots, x_t)$ be a multivariate, non-zero polynomial such that the maximum exponent of any variable is at most d. For any prime power $q > d$, there exists a point $\mathbf{x} \in \mathbb{F}_q^t$ such that $P(\mathbf{x}) \neq 0$.*

Proof. The proof is by induction on t. The base case is when $t = 0$ and therefore P is a polynomial that does not depend on any indeterminates. Since, by assumption, P is not identically zero, the claim holds trivially.

For $t \geq 1$, we may write

$$P(x_1, \ldots, x_t) = \sum_{i=0}^{d} P_i(x_1, \ldots, x_{t-1})\, x_t^i,$$

where at least one of the polynomials P_k must be non-zero. By induction, there is a point $\mathbf{x}' \in \mathbb{F}_q^{t-1}$ such that $P_k(\mathbf{x}') \neq 0$. Substituting $\mathbf{x}'$ for $(x_1, \ldots, x_{t-1})$, P becomes a single-variate, non-zero polynomial in x_t of degree at most d. Since this polynomial can have at most d roots over $\mathbb{F}_q$ and $q > d$, we may choose a value $x_t \in \mathbb{F}_q$ that is not a root. Thus $P(\mathbf{x}', x_t) \neq 0$. ∎

This lemma implies the following simple self-reducibility approach for computing a completion.

Algorithm 3.2: A simple algorithm for finding a completion of a mixed matrix A.

Step 1: Compute the rank r of A using Algorithm 3.1.
Step 2: For each indeterminate x in A
  Step 3: Set $x := 0$.
  Step 4: Compute the rank r' of the resulting matrix using Algorithm 3.1.
  Step 5: If $r' < r$ then set $x := 1$.

Applying Lemma 3.5 with $d = 1$, we see that a completion must exist over any field of size at least 2. Thus for any indeterminate, setting it to either 0 or 1 does not decrease the rank. This simple algorithm can also be interpreted as using the method of conditional probabilities [32] to derandomize the algorithm of Lovász.

3.4.2 A Better Algorithm

It is clear that Algorithm 3.2 is not terribly efficient since it repeatedly invokes Algorithm 3.1 on mixed matrices that are very similar. In this subsection, we present an improved algorithm that requires executing Algorithm 3.1 only once. The idea is to use the independent matching M produced by Algorithm 3.1 to decide which indeterminates are "important" and which are not. The unimportant indeterminates may simply be set to zero, but the important ones require more care.

Let's repeat this discussion more formally. Let A be an $n \times n$ mixed matrix, and let $\tilde{A} = \left(\begin{smallmatrix} I & Q \\ Z & T \end{smallmatrix}\right)$ be the corresponding separated matrix. Recall that the indeterminates in $\mathcal{X}$ correspond to edges in the matroid matching problem G. Let M be a maximum independent matching for G. The ***important indeterminates*** are the ones corresponding to edges used by M, namely the set

$$\mathcal{X}_M \ := \ \{ \ x \in \mathcal{X} \ : \ T_{i,j} = x \text{ for some } (i,j) \in M \ \} \, .$$

We now consider the effect on M of choosing a value for an indeterminate. Since the unimportant indeterminates correspond to edges not chosen by M, it intuitively seems that we can set their value arbitrarily and M will still be a maximum independent matching. This is indeed the case, as we show in Lemma 3.6.

Suppose $T_{n+i,n+j}$ contains some indeterminate x. We will view the operation of setting x to v as actually setting $T_{n+i,n+j}$ to 0 and incrementing $Q_{i,n+j}$ by v. This is equivalent from the point of view of matrix A since $A = Q + T$ (under the implicit translation of the row indices mentioned in Section 3.2). Let $\tilde{A}'$ denote the new separated matrix where Q and T have been modified in this manner, and let G' denote the new matroid matching problem. There are only two differences between G and G'. First, the edge $(n+i, n+j)$ does not appear in G' since x is no longer an indeterminate. Second, modifying the matrix Q may have affected independence of some columns. Thus $\mathbf{M}_U$ (and hence $\mathbf{M}^+$) may have changed somewhat.

Lemma 3.6. *If x is unimportant then setting x to 0 does not decrease the rank.*

Proof. Suppose that $T_{n+i,n+j} = x$ where x is unimportant, i.e., $x \in \mathcal{X} \backslash \mathcal{X}_M$. We claim that M is still a maximum independent matching for G'. First note that assigning x the value 0 does not affect the upper submatrix at all, hence the upper submatrices of $\tilde{A}$ and $\tilde{A}'$ are identical. Thus the edges E_U in G and G' are identical, as are the matroids $\mathbf{M}_U$ and $\mathbf{M}'_U$. The edges E_L in G and G' are identical except that the edge $(n+i, n+j)$ does not appear in G'. Since $(n+i, n+j) \notin M$ by assumption, M is still an independent matching in G'. The rank of $\tilde{A}'$ is at least the cardinality of any independent matching in G', and hence at least the size of M. Since $|M| = 2n$, it follows that $\tilde{A}'$ has full rank. ■

The important indeterminates require more care than the unimportant ones, as illustrated by the following example.

Example 3.7. Consider the matrix $A = \left(\begin{smallmatrix} x_1 & 1 \\ 1 & x_2 \end{smallmatrix}\right)$. The corresponding separated matrix is

$$\tilde{A} = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline z_1 & 0 & x_1 & 0 \\ 0 & z_2 & 0 & x_2 \end{array}\right).$$

An independent matching yields two disjoint sets of columns J_U and J_L that are independent for the upper half and the lower half respectively. In this example, we may have $J_U = \{ \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right) \}$ (the first two columns) and $J_L = \{ \left(\begin{smallmatrix}x_1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\x_2\end{smallmatrix}\right) \}$. The set $\mathcal{X}_M$ would then be $\{x_1, x_2\}$. Setting both x_1 and x_2 to 1 is a bad choice since the original matrix A becomes $\left(\begin{smallmatrix}1&1\\1&1\end{smallmatrix}\right)$, so the rank has decreased. Setting both x_1 and x_2 to 0 is a good choice for this example, but this would have been a poor choice if the original matrix A were $\left(\begin{smallmatrix}x_1&0\\0&x_2\end{smallmatrix}\right)$. ■

Now we explain how to deal with the important indeterminates. Suppose we choose an arbitrary $x \in \mathcal{X}_M$ and assign it a value. We want to argue that this assignment does not decrease the rank, at least for some choices of the value. For unimportant variables, we simply argued that our matching M was still a maximum independent matching for the new matroid matching problem G'. However, if $x \in \mathcal{X}_M$ then x corresponds to some edge in M; after assigning x a value, this edge does not appear in G'. Thus M is not a valid matching for G' and we cannot directly use M to argue that the new separated matrix $\tilde{A}'$ has full rank. Instead, we show in the following lemma that we can construct from M a new maximum independent matching M' for G'.

Lemma 3.8 (Swapping Lemma). *Let x be an indeterminate in $\mathcal{X}_M$ that is in entry $T_{n+i,n+j}$. There exists a value v that we can assign to x such that*

$$M' \;=\; M - (r_{n+i}, c_{n+j}) + (r_{n+i}, c_i) - (\hat{c}_i, c_i) + (\hat{c}_{n+j}, c_{n+j}) \tag{3.6}$$

is a maximum independent matching for the new matroid matching problem G'.

To prove this lemma, we first need a few definitions. Rather than discussing the matching M directly, it is more convenient to discuss the columns (i.e., vertices) in C that are covered by M. Hence we define the following sets of column indices.

$$\begin{aligned} J_U \;&:=\; \{\; c \in C \;:\; (\hat{c}, c) \in M \;\} \\ J_L \;&:=\; \{\; c \in C \;:\; (r, c) \in M \text{ for some } r \in R_L \;\} \end{aligned}$$

These sets of columns satisfy the following basic properties.

1. The columns in J_U are independent for the upper submatrix.

2. The columns in J_L are independent for the lower submatrix.

3. $|J_U| = |J_L| = n$.

4. $J_U \cup J_L = C$.

5. $J_U \cap J_L = \emptyset$.

An equivalent statement of property 2 is as follows: if we consider only the lower half of G (i.e., the subgraph induced by the edges E_L) then J_L must be matchable in this subgraph. We will now use these sets of columns to prove the Swapping Lemma.

Proof (of Lemma 3.8). Let $x \in \mathcal{X}_M$ be the indeterminate in entry $T_{n+i,n+j}$, as in the statement of the theorem. We will prove that we can assign x a value v such that

$$J'_U \ := \ J_U - c_i + c_{n+j} \qquad \text{and} \qquad J'_L \ := \ J_L + c_i - c_{n+j} \tag{3.7}$$

satisfy the basic properties for the matrix $\tilde{A}'$. This is sufficient to prove the statement of the theorem. We break the proof up into the following three claims.

Claim 3.9. *J'_U and J'_L satisfy basic properties 3–5 for the matrix $\tilde{A}'$.*

Proof. Since x is important, $(r_{n+i}, c_{n+j}) \in M$. This implies that $(r_{n+i}, c_i) \notin M$ since the vertex r_{n+i} can have only one incident matching edge. In the lower submatrix, c_i is a column of the diagonal matrix Z, and hence has only one non-zero entry. Therefore there is exactly one edge in E_L that is incident with c_i, namely (r_{n+i}, c_i). We have just argued that this edge is not in M, implying that $c_i \notin J_L$. By choice of x, we have $c_{n+j} \in J_L$. Thus, using basic properties 4 and 5, we have

$$c_i \in J_U \qquad c_{n+j} \notin J_U \qquad c_i \notin J_L \qquad c_{n+j} \in J_L. \tag{3.8}$$

This shows that the set operations in equation (3.7) are all non-trivial, and completes the proof of the claim. $\square$

Claim 3.10. *J'_L satisfies basic property 2 for matrix $\tilde{A}'$.*

Proof. To prove this claim, we will focus only on the lower half of G and modify the matching M so that it becomes a valid matching for the lower half of G'. This new matching will be chosen such that it covers the columns in J'_L. This will show that J'_L is matchable in the lower half of G', which is equivalent to the statement that basic property 2 is satisfied.

We have argued above that $(r_{n+i}, c_{n+j}) \in M$, that (r_{n+i}, c_i) is the only edge incident with c_i in E_L and furthermore this edge is not in M. Thus, considering only the lower half of G', $M - (r_{n+i}, c_{n+j}) + (r_{n+i}, c_i)$ is a matching because we have just matched r_{n+i} with a different neighbor (which was previously uncovered). Since $J'_L = J_L - c_{n+j} + c_i$, the new matching covers J'_L and the claim follows. □

Claim 3.11. *There exists a value v such that J'_U satisfies basic property 1.*

Proof. Recall that R_U is the set of all rows in the upper submatrix. Let A_U denote the submatrix $\tilde{A}[R_U, J_U]$ corresponding to the columns J_U. Since J_U satisfies basic properties 1 and 3, the matrix A_U is non-singular. Let u_i and u_{n+j} be the column vectors in the upper submatrix corresponding to column indices c_i and c_{n+j}. Suppose that we assign x the value v. This effectively adds $v \cdot u_i$ to u_{n+j}, since u_i is the i^{th} elementary vector. In the resulting matrix, let A'_U be the submatrix $\tilde{A}[R_U, J'_U]$ corresponding to the columns J'_U. We may equivalently regard A'_U as the matrix obtained from A_U by removing the column u_i and replacing it with the column $u_{n+j} + v \cdot u_i$. We will show that, for some value v, the matrix A'_U is non-singular. This will establish the proof of the claim.

Note that $\det A'_U = \alpha + v \cdot \beta$, where $\beta = \det(A'_U)_{\text{del}(i,n+j)}$. (This follows from Fact A.8.) We will analyze β by considering the original matrix A_U. Column c_i is an elementary vector with its 1 entry in the i^{th} row. Performing column expansion of $\det A_U$, we delete column c_i and the i^{th} row. The resulting matrix is precisely $(A'_U)_{\text{del}(i,n+j)}$. Since A_U is non-singular, we obtain $0 \neq \det A_U = \pm \det(A'_U)_{\text{del}(i,n+j)}$.

Thus we have shown that $\det A'_U$ is a linear function in v of the form $\alpha + v \cdot \beta$ where $\beta \neq 0$. Thus $-\alpha/\beta$ is the unique value for v that makes A'_U singular. We choose v to be any other value, thereby ensuring that the set J'_U is independent for the upper submatrix of $\tilde{A}'$. Thus, if we assign x the value v, then J'_U will satisfy basic property 1. □

We have shown that J'_U and J'_L satisfy all the basic properties for $\tilde{A}'$, completing the proof. ■

Lemma 3.6 and Lemma 3.8 together tell us how to complete the indeterminates of a mixed matrix. The former allows us to dispense with the unimportant indeterminates, and the latter guarantees that there exists a good value for every important indeterminate. In fact, we can choose the value for an important indeterminate in polynomial time by evaluating $\det A'_U$ symbolically as in Claim 3.11. This leads to a new algorithm for finding a completion of a mixed matrix A over any field. Pseudocode for this algorithm is presented in Algorithm 3.3.

Algorithm 3.3: A better algorithm for finding a completion of a mixed matrix A.

Step 1: Construct $\tilde{A}$ and G.
Step 2: Run Algorithm 3.1 to find a maximum independent matching M.
Step 3: For every $x \notin \mathcal{X}_M$, set $x := 0$.
Step 4: While there exists $x \in \mathcal{X}_M$
    Step 5: Use the Swapping Lemma to find a value v to assign to x.
    Step 6: Remove x from $\mathcal{X}_M$ and set $M := M'$ (as in equation (3.6)).

At the end of this algorithm, all indeterminates have been assigned a value. We regard this algorithm as having set all entries of matrix T to zero and having modified the values of matrix Q. Since the rank of the matrix $\tilde{A}$ never decreases, it follows that the resulting matrix Q has full rank. Therefore the values chosen for $\mathcal{X}$ constitute a completion for the matrix A.

3.4.3 Efficiency of the Better Algorithm

To justify our claim that Algorithm 3.3 is "better" than Algorithm 3.2, we must carefully analyze its runtime. Let the dimensions of the input matrix A be $n \times n$. It is clear that Step 2, which requires $O(n^3 \log n)$ time, dominates the work in Step 1 and Step 3. The loop of Step 4 repeats $O(n)$ times since $|\mathcal{X}_M| \leq |M| = 2n$. The work in Step 6 is clearly negligible.

It remains to analyze the work of Step 5. A straightforward implementation that follows the proof of Claim 3.11 requires $O(n^3)$ time, where the bulk of the work is the determinant computation. We now describe a more efficient implementation that shares work between subsequent invocations. The key idea is to maintain a copy of matrix Q that has been pivoted so that independence tests can be performed quickly. This technique is not new; it has been used in many previous algorithms, including Cunningham's matroid intersection algorithm [8].

Algorithm 3.4: An efficient implementation of Algorithm 3.3.

Step 1: Construct $\tilde{A}$ and G. Let $P := Q$ and let S be the identity.
Step 2: Run Algorithm 3.1 to find a maximum independent matching M.
Step 3: For every $x \notin \mathcal{X}_M$, set $x := 0$.
Step 3b: For each $j \in J_Q$, find an $i \in R_Q$ such that $P_{i,j} \neq 0$ and pivot on $P_{i,j}$.
Step 4: While there exists $x \in \mathcal{X}_M$
  Step 5′: Let $T_{n+i,j}$ be the entry containing x. Set $x := 1 - P_{i,j}$. Pivot on $P_{i,j}$ and update S accordingly.
  Step 6: Remove x from $\mathcal{X}_M$ and set $M := M'$ (see equation (3.6)).

Our efficient implementation will maintain two additional $n \times n$ matrices P and S that satisfy $P = SQ$. The matrix P is obtained from Q by performing Gauss-Jordan pivots, and the matrix S keeps track of these pivots. These matrices will allow us to quickly find appropriate values for the important indeterminates. Pseudocode for this efficient implementation is given in Algorithm 3.4.

We argue correctness of Algorithm 3.4 by proving an invariant. First, we must introduce some notation. In the previous section we defined J_U to be the set of columns that are independent for the upper submatrix. We now need to be more precise. Let J_Q denote the columns of Q that are covered by the matching M and let J_I denote the columns of I that are covered by the matching M. Thus $J_U = J_Q \cup J_I$. Let R_U be the set of all n rows of the upper submatrix. Let $R_I \subseteq R_U$ denote the rows in the upper submatrix where the columns J_I are non-zero, i.e., $r_i \in R_I$ if and only if $c_i \in J_I$. The complementary set of rows is $R_Q := R_U \setminus R_I$. The row indices and column indices of P are taken to be identical to those of Q. In this section, we will use j rather than $n+j$ as a column index for columns in Q and T. The assumed range for index j is accordingly $n + 1 \leq j \leq 2n$.

Invariant 3.12. *When Step 4 is executed, the algorithm has pivoted exactly once on each row in R_Q and each column in J_Q.*

An obvious but useful consequence of this invariant is the following.

Corollary 3.13. *When Step 4 is executed, the submatrix $P[R_Q, J_Q]$ is non-singular and the submatrix $P[R_I, J_Q]$ contains only zeros.*

First we argue that the invariant is initially established by Step 3b. To show that Step 3b can indeed be executed, consider assembling the columns of J_I and J_Q into a single matrix. After a suitable reordering of the rows, we have a matrix of the form $\left(\begin{smallmatrix} I & X \\ 0 & Y \end{smallmatrix}\right)$. (The left columns are those in J_I and the right columns are those in J_Q.) By properties of Gaussian elimination (Fact A.13), we see that Step 3b can always be completed and that the invariant holds afterwards.

How does this invariant allow us to choose a value in Step 5′ quickly? Recall that our goal at this step is to execute the Swapping Lemma efficiently. So, for $c_j \notin J_Q$, we want to understand when c_j can be swapped into J_Q without destroying independence.

Lemma 3.14. *Let c_j be a column of submatrix Q that is not in J_Q. If $P_{i,j} \neq 0$ for some $r_i \in R_I$ then the columns $J_Q + c_j$ of matrix Q are independent.*

Proof. Let $Q' := Q[R_U, J_Q + c_j]$. Our objective is to prove that Q' has full column-rank. To do so, we will analyze an appropriate submatrix of P. Let C_S be the column-set of matrix S and let $S' := S[R_Q + r_i, C_S]$. Setting $P' := S' \cdot Q'$, we obtain

$$P' \ = \ S[R_Q + r_i, C_S] \cdot Q[R_U, J_Q + c_j] \ = \ P[R_Q + r_i, J_Q + c_j].$$

Consider $\det P'$. We apply the block determinant fact (Fact A.10) where $W := P[R_Q, J_Q]$ and $Y := (\ P_{i,j}\ )$. Clearly Y is non-singular, by choice of $P_{i,j}$, and Corollary 3.13 shows that the matrix W is non-singular. Therefore, the block determinant fact shows that P' is also non-singular. Since P' has full rank and rank cannot increase by multiplication (Fact A.12), we see that Q' also has full column rank. ■

We now show that the hypothesis of Lemma 3.14 will hold if we choose an appropriate value v for x.

Lemma 3.15. *Let x be the indeterminate in $T_{n+i,j}$. If x is assigned the value $v = 1 - P_{i,j}$ then $P_{i,j}$ becomes non-zero. Furthermore, $r_i \in R_I$.*

Proof. First, consider what happens to the various matrices when we assign x a value. Setting x to the value v amounts to setting $T_{n+i,j} := 0$ and $Q_{i,j} := Q_{i,j} + v$. To determine how the matrix P is affected, recall that $P = SQ$. The column view of matrix multiplication (see Fact A.1) tells us that $P_{*,j} = \sum_c S_{*,c} Q_{c,j}$. Therefore incrementing $Q_{i,j}$ by v results in incrementing $P_{*,j}$ by $v \cdot S_{*,i}$. Thus whether $P_{i,j}$ becomes non-zero depends on the value of $S_{i,i}$.

To understand the value of $S_{i,i}$, we first show that $r_i \in R_I$. This holds because it was shown in Lemma 3.8 that if $T_{n+i,j}$ contains an indeterminate in $\mathcal{X}_M$ then $c_i \in J_I$ (see equation (3.8)). Thus, by definition, we have that $r_i \in R_I$. Therefore the invariant shows that we have not yet pivoted on row i. By the properties of Gaussian elimination (Fact A.15), if we have not yet pivoted on row i then the i^{th} column of S is the i^{th} elementary vector and hence $S_{i,i} = 1$. Therefore incrementing $Q_{i,j}$ by v results in incrementing $P_{i,j}$ by v. Choosing $v = 1 - P_{i,j}$ ensures that $P_{i,j}$ becomes non-zero. ■

This lemma shows that the choice of v in Algorithm 3.4 makes $P_{i,j}$ non-zero, and therefore Step 5′ can indeed pivot on $P_{i,j}$ as specified. To complete the proof of correctness, we must argue two remaining points: (1) that Invariant 3.12 is restored, and (2) the new matching M' is a maximum independent matching for G'. For the first point, note that setting the matching to M' has the effect of setting $J_Q := J_Q + c_j$ and $R_Q := R_Q + r_i$. Since the algorithm has just pivoted on entry $P_{i,j}$ this shows that the invariant is restored. To prove the second point, we will show that the Swapping Lemma holds with our particular value of v by revising Claim 3.11.

Claim 3.16. *Setting $v := 1 - P_{i,j}$ ensures that J'_U satisfies basic property 1.*

Proof. Combine the columns $J_I - c_i$ and $J_Q + c_j$ into a single matrix, which we denote $M_{U'}$. We may reorder the rows such that the matrix has the form $M_{U'} = \left(\begin{smallmatrix} W & X \\ 0 & Y \end{smallmatrix}\right)$, where $W = I[R_I - r_i, J_I - c_i]$ and $Y = Q[R_Q + r_i, J_Q + c_j]$. It suffices to show that $M_{U'}$ is non-singular. Clearly submatrix W is non-singular, so consider submatrix Y. The algorithm has so far pivoted exactly once in each row of $R_Q + r_i$ and each column of $J_Q + c_j$. By properties of Gaussian elimination (Fact A.14), we see that $Y = Q[R_Q + r_i, J_Q + c_j]$ is non-singular. The block determinant fact now implies that $M_{U'}$ is non-singular. ■

The correctness of Algorithm 3.4 has been established. We now analyze its runtime by considering the work of Step 3b and Step 5′. It suffices to consider the pivoting operations. Since the algorithm pivots at most once in each column, there are at most n pivot operations. Since each pivot requires only $O(n^2)$ time (see Appendix A), the pivots require only $O(n^3)$ time in total. Thus the running time of the algorithm is dominated by Step 2, which requires $O(n^3 \log n)$ time. This proves the first claim of Theorem 1.2.

Let us now compare Algorithm 3.4 with existing algorithms for finding completions. The randomized algorithm of Lovász requires $O(n^3)$ time to verify the rank of the resulting matrix and, in order to have a reasonable probability of success, operates over a field of size $\Omega(n)$. Geelen's deterministic algorithm can be implemented in $O(n^4)$ time [3] and also operates over a field of size $\Omega(n)$. In comparison, our algorithm is only a logarithmic factor slower than the randomized algorithm, completes successfully with certainty, and can operate over any field. A slight difference between Geelen's algorithm and ours is that Geelen's does not allow indeterminates to be assigned the value zero. Our algorithm can be adapted to include this requirement by operating over any field with at least three elements. Details are provided in Section 3.5.1.

3.5 Simultaneous Matrix Completion

Let $\mathcal{A} := \{A_1, \ldots, A_d\}$ be a collection of mixed matrices over $\mathbb{F}_q$, where each indeterminate may appear in several matrices, but at most once in any particular matrix. Let $\mathcal{X}$ be the set of all indeterminates appearing in the matrices in $\mathcal{A}$. In this section, we consider the problem of finding a simultaneous max-rank completion for the matrices in $\mathcal{A}$. That is, we seek an assignment of values to the indeterminates in $\mathcal{X}$ from $\mathbb{F}_q$ such that the resulting matrices all have full rank. In this section we prove that if $q > d$ then such a completion must exist and we give an efficient algorithm for finding such a completion.

The existence result is a straightforward extension of the result for completion of a single matrix. Define $P := \prod_{k=1}^{d} \det A_k$. Then P is a multivariate polynomial in the indeterminates $\mathcal{X}$ where the maximum exponent of each variable is d. Lemma 3.5 shows that for any $q > d$, there is a point $\mathbf{x} \in \mathbb{F}_q^{|\mathcal{X}|}$ such that $P(\mathbf{x}) \neq 0$. That is, there exists an assignment of values from $\mathbb{F}_q$ to the indeterminates in $\mathcal{X}$ such that the resulting matrices all have full rank. Furthermore, this existence result immediately implies a polynomial-time algorithm to find a simultaneous completion, using the same self-reducibility approach as Algorithm 3.2. Simply pick an indeterminate and repeatedly try assigning it values from $\mathbb{F}_q$ until all matrices have full-rank. Repeat this process until all indeterminates have been assigned a value.

We now derive an improved algorithm by extending Algorithm 3.4 to handle multiple matrices. The intuitive idea is, rather than blindly guessing values for an indeterminate, maintain pivoted copies of each matrix so that we can quickly find a value which will not decrease its rank. The first step is to construct the separated matrix $\tilde{A}_k = \left(\begin{smallmatrix} I_k & Q_k \\ Z_k & T_k \end{smallmatrix}\right)$ associated with A_k for all k. For simplicity of notation, we will now focus on a single matrix $\tilde{A}$ and drop the subscript k. Suppose we execute Algorithm 3.1 on $\tilde{A}$ and obtain an independent matching M. Let J_U and $\mathcal{X}_M$ be as defined in the previous section.

Lemma 3.17. *Let x be an arbitrary indeterminate in $\mathcal{X}$. There is at most one $v \in \mathbb{F}_q$ such that assigning x the value v decreases the rank of $\tilde{A}$.*

Proof. If x does not appear in $\tilde{A}$ then there is nothing to prove, so assume that some entry $T_{n+i,j}$ contains x.

Case 1: $x \in \mathcal{X}_M$. The argument proceeds as in Lemma 3.8. If we assign x the value v then we must ensure that the set of vectors $J'_U := J_U - c_i + c_j$ is linearly independent. As in Claim 3.11, we observe that $\det A'_U$ has the form $\alpha + v \cdot \beta$ where $\beta \neq 0$. Thus there is precisely one value of v that makes J'_U linearly dependent.

Case 2: $j \in J_U$. Note that in this case j must be a column of the submatrix Q. Then setting x to v amounts to incrementing $Q_{i,j}$ by v. Let A_U be the matrix consisting of columns in J_U and let A'_U be the matrix that results from incrementing by v. Note that $\det A'_U = \alpha + v \cdot \beta$, where $\alpha = \det A_U \neq 0$. (This follows from Fact A.8.) Thus $\det A'_U$ can only be zero when $\beta \neq 0$ and $v = -\alpha/\beta$. This shows that there is at most one value of v such that $\det A'_U = 0$.

Case 3: $j \notin J_U$ and $x \notin \mathcal{X}_M$. As in Lemma 3.6, assigning a value to x does not affect independence of J_U, nor does it destroy any edges used by the matching. Thus the rank of $\tilde{A}$ is unaffected. ∎

Lemma 3.17 implies the following approach for finding a simultaneous completion for a set of matrices $\mathcal{A}$. Pick an indeterminate x and, for each matrix in $\mathcal{A}$, determine the forbidden value of x, as in the proof of Lemma 3.17. Since we operate over $\mathbb{F}_q$ and $q > d$, a non-forbidden value must exist. Assigning this value to x preserves the rank of all matrices. Pseudocode for this simultaneous completion

Algorithm 3.5: Algorithm for finding a simultaneous max-rank completion of the matrices in the set $\mathcal{A}$.

Step 1: For k from 1 to d,

Step 2: Consider $A = A_k$. Construct the separated matrix $\tilde{A} = \left(\begin{smallmatrix} I & Q \\ Z & T \end{smallmatrix}\right)$ associated with A. Compute a maximum independent matching M using Algorithm 3.1. Initially let $P := Q$ and let S be the identity matrix, so we have $P = SQ$. The objects $\tilde{A}$, M, P etc. are implicitly parameterized by k.

Step 3: Define $\mathcal{X}_M$, J_Q and R_Q as in Section 3.4.2 and Section 3.4.3. For each $j \in J_Q$, find an $i \in R_Q$ such that $P_{i,j} \neq 0$ and pivot on $P_{i,j}$.

Step 4: For each $x \in \mathcal{X}$,

Step 5: The set of forbidden values for x is initially $F := \emptyset$.

Step 6: For k from 1 to d,

Step 7: Consider $A = A_k$. Let $T_{n+i,j}$ be the entry of A containing x. We determine A's forbidden value for x as in Lemma 3.17. If $x \in \mathcal{X}_M$ then, as in Algorithm 3.4, $-P_{i,j}$ is the forbidden value for x. Set $F := F \cup \{-P_{i,j}\}$.

Step 8: Otherwise, suppose that $j \in I_Q$. Setting $x := v$ results in incrementing $Q_{i,j}$ by v and incrementing $P_{*,j}$ by $v \cdot S_{*,i}$ (by the column-based view of matrix multiplication). Let i' be the row such that $P_{i',j} \neq 0$. We must ensure that this entry remains non-zero after incrementing. Store the value of i' for use in Step 11. Setting $x := v$ results in increasing $P_{i',j}$ by $v \cdot S_{i',i}$. (Other entries of $P_{*,j}$ may also be modified but they will be zeroed in Step 11.) If $S_{i',i} \neq 0$ then the value $-P_{i',j}/S_{i',i}$ is forbidden. Set $F := F \cup \{-P_{i',j}/S_{i',i}\}$.

Step 9: Choose a value v from $\mathbb{F}_q$ that is not in F. Assign the value v to x.

Step 10: For k from 1 to d,

Step 11: Consider $A = A_k$. Update T, Q and P according to the value v for x. If $x \in \mathcal{X}_M$, update $\mathcal{X}_M$ and M as in Step 6 of Algorithm 3.4. Otherwise, if $j \in J_Q$, pivot on entry $P_{i',j}$.

algorithm is given in Algorithm 3.5.

We now briefly consider the running time of this algorithm. Assume for simplicity that all matrices have size $n \times n$. The time required for Steps 1–3 is clearly $O(dn^3 \log n)$ since the work is dominated by the d invocations of Algorithm 3.1. The time for Steps 4–11 is dominated by the pivoting work in Step 11. For each indeterminate in $\mathcal{X}$ and for each matrix we perform at most one pivot, which requires $O(n^2)$ time. Thus the time required for Steps 4–11 is $O(|\mathcal{X}|\, dn^2)$. Thus the total runtime of this algorithm is $O(d(n^3 \log n + |\mathcal{X}|\, n^2))$. This discussion completes the proof of Theorem 1.3.

3.5.1 Column Compatibility

The performance of the simultaneous matrix completion algorithm may be slightly improved for collections of matrices satisfying a certain technical property. The general approach is to reduce the number of pivoting operations by taking advantage of indeterminates that appear together in the same column in each matrix. Such indeterminates may all be assigned a value simultaneously before performing any pivoting work.

Formally, define an equivalence relation by the transitive closure of the relation $x_i \sim x_j$ if x_i and x_j appear in the same column in some matrix in $\mathcal{A}$. The collection $\mathcal{A}$ is called ***column-compatible*** if, for every matrix A_k and equivalence class $[x]$, at most one column of A_k contains indeterminates in $[x]$. The column-compatibility condition deserves consideration because the condition is satisfied by the matrices arising from multicast network coding problems.

For column-compatible collections of matrices, Steps 4–11 of Algorithm 3.5 may be modified as follows. We repeatedly choose an equivalence class $[x]$, find values for each indeterminate in that class that preserve the rank of all matrices, then, pivot each matrix on the column (if any) that contains indeterminates in $[x]$. This approach improves the time required for Steps 4–11 to $O(|\mathcal{X}|\, dn + dn^3)$, so the total runtime of the algorithm becomes $O(d(n^3 \log n + |\mathcal{X}|\, n))$.

As mentioned in the previous section, Geelen's algorithm for finding a completion of a single mixed matrix does not allow indeterminates to be assigned the value zero. Such a completion can also be found using a slight variant of Algorithm 3.5. Given a mixed matrix A, we define a singleton collection $\mathcal{A} := \{A\}$. The only change required to Algorithm 3.5 is to forbid the value 0 by initializing $F := \{0\}$ rather than $F := \emptyset$ in Step 5. This algorithm may operate over $\mathbb{F}_3$ or any larger field, and the time required is $O(n^3 \log n)$ since a singleton collection is trivially column-compatible.

3.6 Single Matrix Completion, Revisited

In this section we describe our last algorithm for finding a completion of a single mixed matrix. This algorithm depends on a fairly complicated matroid-theoretic algorithm, but is otherwise extremely simple. Pseudocode for this algorithm is shown in Algorithm 3.6. The discussion of this algorithm was postponed to the present section because it does not extend to simultaneous matrix completions.

Algorithm 3.6: A very simple algorithm for finding a completion of a mixed matrix A.

Step 1: Construct $\tilde{A}$ and G.
Step 2: Find a maximum independent matching M in G that minimizes $|\mathcal{X}_M|$.
Step 3: Assign $x := 1$ if $x \in \mathcal{X}_M$ and $x := 0$ otherwise.

Given a mixed matrix A, Step 1 constructs the separated matrix $\tilde{A} = \left(\begin{smallmatrix} I & Q \\ Z & T \end{smallmatrix}\right)$ and the matroid matching problem G as in Section 3.4. As before, $\mathcal{X}$ denotes the indeterminates appearing in A and $\mathcal{X}_M \subseteq \mathcal{X}$ denotes the set of indeterminates that correspond to edges in a matching M. Additionally, define $\mathcal{Z}$ to be the set of new indeterminates that are contained in the diagonal submatrix Z and define $\mathcal{Z}_M$ analogously. Step 2 of this new algorithm finds a maximum independent matching M satisfying a certain condition. To explain why such a matching is useful, we must examine $\det \tilde{A}$ in some detail. This determinant consists of a linear combination of monomials of the form

$$x_{\alpha_1} x_{\alpha_2} \cdots x_{\alpha_k} z_{\beta_1} z_{\beta_2} \cdots z_{\beta_{n-k}}.$$

We first show a connection between the monomials in $\det \tilde{A}$ and matchings in G.

Lemma 3.18. *There is a bijection between the maximum independent matchings in G and the monomials that appear in $\det \tilde{A}$ such that a matching M is mapped to a monomial whose indeterminates are precisely those in $\mathcal{X}_M \cup \mathcal{Z}_M$.*

Proof. The bijection is based on the Laplace expansion of $\det \tilde{A}$ appearing in equation (3.5). To prove the forward direction, consider a matching M. Let J be the set of columns that are covered by edges of M in the lower half of G. Note that J is the set of columns containing the indeterminates in $\mathcal{X}_M$. Since M is an independent matching, the columns $\overline{J}$ are independent for the upper submatrix. Thus, if we consider the term in the sum of equation (3.5) with this value of J, we see that $\det U[R_U, \overline{J}] \neq 0$. Furthermore, since each indeterminate appears at most once in A, there is no cancellation among terms of the determinant and hence $\det L[R_L, J]$ contains a monomial whose indeterminates are precisely those in $\mathcal{X}_M \cup \mathcal{Z}_M$. Thus

$\det L[R_L, J] \cdot \det U[R_U, \overline{J}]$ contains this monomial and so does $\det \tilde{A}$. The other direction of the bijection follows by applying same construction in reverse. ■

Let us now consider the matching M chosen in Step 2 of Algorithm 3.6.

Lemma 3.19. *Every monomial in* $\det \tilde{A}$ *except one contains an indeterminate in* $\mathcal{X} \setminus \mathcal{X}_M$.

Proof. We have shown in Lemma 3.18 that there is a monomial corresponding to M that contains only indeterminates in $\mathcal{X}_M \cup \mathcal{Z}_M$. Consider any other monomial that does not contain an indeterminate in $\mathcal{X} \setminus \mathcal{X}_M$. Let M' be the matching corresponding to this monomial. Our choice of M implies that $|\mathcal{X}_{M'}| \geq |\mathcal{X}_M|$. Since $\mathcal{X}_{M'}$ does not contain an indeterminate in $\mathcal{X} \setminus \mathcal{X}_M$, we must have $\mathcal{X}_{M'} = \mathcal{X}_M$. Thus the edges of M' corresponding to indeterminates in $\mathcal{X}$ are identical to those in M. Let us consider the edges of M' in the lower half of G that correspond to indeterminates in $\mathcal{Z}$. Since the matrix Z is diagonal, there is only one possibility for these edges. Thus we conclude that $M = M'$ and hence there is a unique monomial that does not contain an indeterminate in $\mathcal{X} \setminus \mathcal{X}_M$. ■

Suppose that we assign the value 1 to all indeterminates in $\mathcal{X}_M \cup \mathcal{Z}$ and 0 to all indeterminates in $\mathcal{X} \setminus \mathcal{X}_M$. It follows from Lemma 3.19 that exactly one monomial will have the value 1 and that all others will have the value 0. This assignment therefore ensures that $\det \tilde{A} \neq 0$ and hence gives a completion for $\tilde{A}$ and for the original matrix A. This discussion proves the correctness of Algorithm 3.6.

The running time of this algorithm is obviously dominated by Step 2. To find such a matching M, we augment the matroid matching problem G with weights. Edges corresponding to indeterminates in $\mathcal{X}$ are given weight 0 and all other edges are given weight 1. A maximum weight independent matching therefore gives the desired matching M. As described in Appendix B, such a matching can be computed in $O(n^3 \log^2 n)$ time. This bound reduces to $O(n^{2.77})$ if fast matrix multiplication is used. This completes the proof of Theorem 1.2.

Chapter 4

Hardness of Matrix Completion

The previous chapter showed a polynomial time algorithm that finds a completion for a single matrix over *any* field. We now show that finding a simultaneous matrix completion is computationally hard unless the field size is sufficiently large.

4.1 Simultaneous Completion when $q < d$

Define SIM-COMPLETION(q, d) to be the problem of deciding whether a given collection $\mathcal{A}$ of d matrices over $\mathbb{F}_q$ has a simultaneous completion. Recall that q must be a prime power in order for $\mathbb{F}_q$ to be a field.

Theorem 4.1. *If q is a prime power and $d > q$, the problem* SIM-COMPLETION(q, d) *is NP-complete.*

Note that fixing q and d does not determine the size of the input, because the matrices can be arbitrarily large.

Proof. First we remark that a completion can be checked in polynomial time. Simply perform Gaussian elimination on the completed matrices and verify that they have full rank. The Gaussian elimination process runs in polynomial time because the matrices no longer contain any indeterminates; they only contain values in $\mathbb{F}_q$. This shows that the problem is in NP. We now show hardness via a reduction from CIRCUIT-SAT.

The CIRCUIT-SAT problem can be defined as follows: given an acyclic boolean circuit of NAND gates with a single output, determine if the inputs can be set such that the output becomes 1. Suppose we are given a circuit ϕ with n gates. We will construct a collection $\mathcal{A}$ of d matrices over $\mathbb{F}_q$ such that a completion for $\mathcal{A}$ corresponds to a satisfying assignment for ϕ and vice-versa. It suffices to consider the case that $d = q + 1$ because otherwise we may simply pad the set $\mathcal{A}$ with additional non-singular matrices containing no variables. Suppose initially that the field size is $q = 2$, i.e., all values are either 0 or 1.

Gadget 1: The first step is the construction of a matrix gadget that behaves like a NAND gate. Consider a completion of the mixed matrix

$$N(a,b,c) \ := \ \begin{pmatrix} 1 & a \\ b & c \end{pmatrix}.$$

Recall that a completion must maximize the rank of this matrix and therefore must satisfy $0 \neq \det N(a,b,c) = c - ab$. The only non-zero element of $\mathbb{F}_2$ is 1, hence $c = ab + 1$. Since multiplication over $\mathbb{F}_2$ corresponds to the boolean AND operation and adding 1 corresponds to boolean NOT, we have $c = a$ NAND b.

It is now straightforward to construct a (large) collection of matrices for which a simultaneous completion corresponds to a satisfying assignment for ϕ. For every NAND gate in ϕ with inputs a and b and output c, add an instance of $N(a,b,c)$ to $\mathcal{A}$. We also add to $\mathcal{A}$ the matrix $(\, y \,)$ where y is the output of the circuit. This reflects the requirement that the circuit be satisfied, i.e., its output should be 1. It is easy to verify that a completion for $\mathcal{A}$ is a satisfying assignment for ϕ and vice-versa. Unfortunately, this collection of matrices is too large: it has $|\mathcal{A}| = n + 1$ matrices and we desire that $|\mathcal{A}| = d = q + 1 = 3$.

To construct a smaller collection of matrices, we might imagine combining these matrices into a single block-diagonal matrix A. For example, we might have

$$A \ = \ \begin{pmatrix} N(a,b,c) & & & \\ & N(b,c,d) & & \\ & & N(a,d,e) & \\ & & & \ddots \end{pmatrix}.$$

Unfortunately such a matrix is not a valid input for the simultaneous matrix completion problem since variables appear multiple times in general. The next gadget allows us to circumvent this restriction.

Gadget 2: We now construct a gadget that allows us to replicate variables. Consider the following pair of matrices.

$$\begin{aligned} B &:= \begin{pmatrix} N(1,x^{(1)},y^{(2)}) & & & \\ & N(1,x^{(2)},y^{(3)}) & & \\ & & N(1,x^{(3)},y^{(4)}) & \\ & & & \ddots \end{pmatrix} \\ C &:= \begin{pmatrix} N(1,y^{(2)},x^{(2)}) & & & \\ & N(1,y^{(3)},x^{(3)}) & & \\ & & N(1,y^{(4)},x^{(4)}) & \\ & & & \ddots \end{pmatrix} \end{aligned} \tag{4.1}$$

The matrix B enforces that $y^{(i+1)} = 1 \text{ NAND } x^{(i)} = \text{NOT } x^{(i)}$ for all $i \geq 1$. The matrix C similarly enforces that $x^{(i)} = \text{NOT } y^{(i)}$ for all $i \geq 2$. Thus B and C together enforce that $x^{(i)} = x^{(1)}$ for all $i \geq 1$. We extend the definition of B and C to generate n copies of all variables in ϕ (including the inputs and intermediate wires).

Construct the matrix A' by modifying A so that the first appearance of a variable x uses $x^{(1)}$, the second appearance uses $x^{(2)}$, etc. Let $\mathcal{A} = \{A', B, C\}$. It is easy to verify that a simultaneous matrix completion for $\mathcal{A}$ determines a satisfying assignment for ϕ and vice-versa. This completes the proof for the special case of $q = 2$. Our third gadget is needed to generalize to an arbitrary field $\mathbb{F}_q$.

Gadget 3: This third gadget ensures that each variable can only be assigned the value 0 or 1. Let c be a fixed non-zero constant in $\mathbb{F}_q$. Consider a completion of the mixed matrix

$$D_c := \begin{pmatrix} c & c & & & \\ c & x_1 & & & \\ & & c & c & \\ & & c & x_2 & \\ & & & & \ddots \end{pmatrix}.$$

(Here c is not an indeterminate that requires completion, it is a parameter of the matrix D_c.) If D_c is to have full rank, it is clear that none of the indeterminates can be assigned the value c. Furthermore, if all indeterminates are assigned values in $\{0,1\}$ then D_c has full rank. By adding the matrices $\{D_2,\ldots,D_{q-1}\}$ to the collection $\mathcal{A}$, we enforce that each indeterminate is either 0 or 1.

We now verify that the first two gadgets behave as desired when operating over $\mathbb{F}_q$ but the indeterminates are restricted to the values 0 and 1. It suffices to consider the matrix $N(a,b,c)$. Recall that $\det N(a,b,c) = c - ab$. If a,b,c are elements of $\mathbb{F}_q$ that are restricted to take on the values $\{0,1\}$ then the set of solutions to the equation $c - ab \neq 0$ is precisely

$$(a,b,c) \in \{(0,0,1),(0,1,1),(1,0,1),(1,1,0)\},$$

which again corresponds to the NAND operation. This verifies that our third gadget indeed reduces the case of general q to the case $q=2$.

In summary, $\mathcal{A} = \{A', B, C, D_2, \ldots, D_{q-1}\}$ is a collection of $q+1$ matrices over $\mathbb{F}_q$ for which a completion corresponds to a satisfying assignment for ϕ and vice-versa. Given ϕ, the collection $\mathcal{A}$ is easy to construct in polynomial time, and hence SIM-COMPLETION(q,d) is NP-hard. ■

As a special case of Theorem 4.1, we see that deciding whether three matrices over $\mathbb{F}_2$ have a simultaneous completion is NP-complete. What about two matrices over $\mathbb{F}_2$? A common phenomenon in theoretical computer science is that the difference between 2 and 3 often means the difference between polynomial-time decidability and NP-hardness. (See, for example, Papadimitriou [40].) A well-known example is obtained by restricting the clause sizes of a CNF formula: 3SAT is NP-hard and 2SAT is polynomial-time decidable. Another example is coloring the vertices of a graph: 3-coloring is NP-hard whereas 2-coloring is easy. Yet another example is obtained by restricting the number of appearances of a variable in a CNF formula: 3SAT is NP-hard even if each variable appears at most three times, but SAT is polynomial-time decidable if each variable appears at most twice.

One might conjecture that this 2-or-3 phenomenon extends to simultaneous matrix completions, since the number of appearances of a variable in a set of matrices seems related to the number of appearances of a variable in a CNF formula. In fact, our proof of Theorem 4.1 bears strong similarity to the proof that 3SAT is NP-complete when each variable appears at most three times. Can we show analogously that completing two matrices over $\mathbb{F}_2$ is easy? Somewhat surprisingly, we will show in Section 4.2 that the opposite result is true: SIM-COMPLETION$(2,2)$ is NP-complete.

Before doing so, we use rudimentary polyhedral combinatorics to prove a result claimed above. To the best of our knowledge, this proof was not previously known.

Theorem 4.2 (Tovey [46]). SAT *is decidable in polynomial time if each variable appears at most twice.*

Proof. Suppose we are given a CNF formula ϕ where each variable appears at most twice. If both appearances of a variable are unnegated then clearly we can assume that this variable is true and delete the clauses that contain it. A similar argument holds if both appearances of a variable are negated, hence we may assume that a variable appears negated exactly once and unnegated exactly once.

Let m be the number of clauses in ϕ and let $x_1, \ldots, x_n$ be the variables. Define

$$\begin{aligned} A_i &:= \{\, j \,:\, x_j \text{ appears unnegated in clause } i \,\} \\ B_i &:= \{\, j \,:\, x_j \text{ appears negated in clause } i \,\}. \end{aligned}$$

The linear programming relaxation of the satisfiability problem is a feasibility problem for the following polytope P:

$$\begin{aligned} &\sum_{j \in A_i} x_j \;+\; \sum_{j \in B_i} (1 - x_j) \;\geq\; 1 && (\text{for } 1 \leq i \leq m) \\ &0 \;\leq\; x_j \;\leq\; 1 && (\text{for } 1 \leq j \leq n) \end{aligned}$$

Any integer vector in this polytope corresponds to a satisfying assignment for ϕ and vice-versa. So it suffices to find an integer vector in this polytope. The first m constraints of this polytope may be written as the system of equations $M\mathbf{x} \geq \mathbf{c}$. Recall our previous assumption that each variable appears once negated and once unnegated. This implies that each column of M contains exactly two non-zero entries, one being $+1$ and the other being -1. Such matrices are well known to be ***totally unimodular*** [44, Theorem 13.9], and hence the polytope P has integer vertices [31, Theorem 7C.1]. If the formula ϕ is satisfiable then P is non-empty and we can find an integer vertex in polynomial time via the ellipsoid method. This vertex directly yields a satisfying assignment for ϕ. ∎

4.2 Simultaneous Completion when $q = d$

Example 4.3. Consider the following two matrices over $\mathbb{F}_2$.

$$M_0 := \begin{pmatrix} 1 & 0 \\ 0 & x \end{pmatrix} \qquad \text{and} \qquad M_1 := \begin{pmatrix} 1 & 1 \\ 1 & x \end{pmatrix}.$$

These matrices do not have a simultaneous completion because M_0 is singular when $x = 0$ and M_1 is singular when $x = 1$. This example easily generalizes to show that there exist q matrices over a field $\mathbb{F}_q$ that do not have a simultaneous completion. To do so, we simply use the matrices encountered earlier which force all values to be 0 or 1, namely $M_c := \left(\begin{smallmatrix} c & c \\ c & x \end{smallmatrix}\right)$ for $2 \leq c < q$ ■

Recall Algorithm 3.2, the simple self-reducibility algorithm for finding a completion of a single matrix over $\mathbb{F}_2$. The algorithm chooses an indeterminate arbitrarily, assigns it a value arbitrarily, and verifies that the matrix still has full rank. One might imagine generalizing this algorithm to find a simultaneous completion for two matrices over $\mathbb{F}_2$: assign values to indeterminates arbitrarily and verify that *both* matrices still have full rank. The following example shows that this generalization does not work.

Example 4.4. We show that the generalization of Algorithm 3.2 described above will err on the following two matrices.

$$A := \begin{pmatrix} x & 0 & 1 \\ y & 1 & 0 \\ z & 1 & 1 \end{pmatrix} \qquad \text{and} \qquad B := \begin{pmatrix} x & 0 & 1 \\ y & 1 & 0 \\ 0 & z & 1 \end{pmatrix}$$

These matrices have completions when the following equations are satisfied.

$$\begin{aligned} \det A &= x + y + z = 1 & \implies \quad & (x, y, z) \in \{(1,0,0), (0,1,0), (0,0,1), (1,1,1)\} \\ \det B &= x + yz = 1 & \implies \quad & (x, y, z) \in \{(1,0,0), (1,0,1), (1,1,0), (0,1,1)\} \end{aligned}$$

The only simultaneous completion for these two matrices is $(x, y, z) = (1, 0, 0)$. Consider now the operation of Algorithm 3.2 on these matrices. It begins by arbitrarily setting the indeterminate x to 0. The resulting matrices are both still non-singular because their determinants are $\det A = y + z$ and $\det B = yz$ respectively. Next, the indeterminate y is considered. Setting y to 0 makes B singular so y is set to 1. The resulting determinants are $\det A = 1 + z$ and $\det B = z$. However, there is no choice of z that ensures both determinants are non-zero, and hence the algorithm fails. ■

The preceding example shows that a simple algorithm fails to find a completion for two matrices over $\mathbb{F}_2$. This failure is to be expected — we will show that SIM-COMPLETION(q, q) is NP-complete for every prime power $q \geq 2$. The key to proving this result is the following lemma.

Lemma 4.5. *For any $n \geq 1$, define the mixed matrix*

$$R_n(x_1, \ldots, x_n) \; := \; \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 & 0 \\ x_1 & 1 & 1 & \ldots & 1 & 1 \\ 0 & x_2 & 1 & \ldots & 1 & 1 \\ 0 & 0 & x_3 & \ldots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & x_n & 1 \end{pmatrix}.$$

The determinant of this matrix (over any field) is

$$\det R_n \; = \; \prod_{i=1}^{n}(1 - x_i) \; - \; \prod_{i=1}^{n}(-x_i). \tag{4.2}$$

Proof. Construct the following matrices from R_n by modifying the last column.

$$R'_n \; := \; \begin{pmatrix} 1 & \ldots & 1 & 1 \\ x_1 & \ldots & 1 & 1 \\ 0 & \ddots & \vdots & \vdots \\ 0 & \ldots & x_n & 1 \end{pmatrix} \qquad \text{and} \qquad R''_n \; := \; \begin{pmatrix} 1 & \ldots & 1 & -1 \\ x_1 & \ldots & 1 & 0 \\ 0 & \ddots & \vdots & \vdots \\ 0 & \ldots & x_n & 0 \end{pmatrix}$$

Since the determinant is a linear function of every column (Fact A.7), we have $\det R_n = \det R'_n + \det R''_n$. The latter determinant is easily computed by the block determinant fact.

$$\det R_n'' = (-1)^n \cdot \det \begin{pmatrix} -1 \end{pmatrix} \cdot \det \begin{pmatrix} x_1 & \dots & 1 \\ 0 & \ddots & \vdots \\ 0 & \dots & x_n \end{pmatrix} = -\prod_{i=1}^{n} (-x_i) \tag{4.3}$$

To compute $\det R_n'$, we will use the Schur complement. Moving the first row to the last row, we incur a sign change of $(-1)^n$ and obtain the block matrix $\left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$, where

$$W = \begin{pmatrix} x_1 & \dots & 1 \\ 0 & \ddots & \vdots \\ 0 & \dots & x_n \end{pmatrix}, \qquad X = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \qquad Y = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 \end{pmatrix}.$$

The properties of the Schur complement (Fact A.11) now imply that

$$\begin{aligned} \det R_n' &= (-1)^n \cdot \det Z \cdot \det \left(W - XZ^{-1}Y \right) \\ &= (-1)^n \cdot \det \left(W - XY \right) \\ &= (-1)^n \cdot \det \left(\begin{pmatrix} x_1 & 1 & \dots & 1 \\ 0 & x_2 & \dots & 1 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & x_n \end{pmatrix} - \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \right) \\ &= (-1)^n \cdot \det \begin{pmatrix} x_1 - 1 & 0 & \dots & 0 \\ -1 & x_2 - 1 & \dots & 0 \\ -1 & -1 & \ddots & \vdots \\ -1 & -1 & \dots & x_n - 1 \end{pmatrix} \\ &= \prod_{i=1}^{n} (1 - x_i). \end{aligned}$$

Thus we have shown that $\det R_n = \det R_n' + \det R_n'' = \prod_{i=1}^{n}(1 - x_i) - \prod_{i=1}^{n}(-x_i)$. ■

Lemma 4.5 can also be proven by induction, but we feel that our purely linear algebraic proof better captures the structure of the matrix R_n.

Theorem 4.6. *If q is a prime power, the problem* SIM-COMPLETION(q, q) *is NP-complete.*

Proof. This theorem is an adaption of Theorem 4.1, where the replication gadget is based on the matrix R_n rather than the matrices B and C (cf., equation (4.1)).

Suppose we are given ϕ, an instance of CIRCUIT-SAT. We construct the matrices A' and D_c $(2 \leq c \leq q-1)$ as in Theorem 4.1. Recall that A' is a block-diagonal matrix with a block for each NAND gate, where each appearance of a variable uses a distinct replicated copy.

Next we will use the matrix R_n. Lemma 4.5 shows that $\det R_n$ is the arithmetization of the boolean formula

$$\bigwedge_{i=1}^{n} \overline{x_i} \;\vee\; \bigwedge_{i=1}^{n} x_i.$$

Thus the matrix R_n satisfies the following curious property: any completion for R_n where $x_1, \ldots, x_n \in \{0,1\}$ satisfies $x_1 = \cdots = x_n$. This is precisely the property needed to replicate a variable n times. Thus, for every variable (including the inputs and intermediate wires) $a, b, c, \ldots$ in ϕ, we add a copy of R_n to the following matrix.

$$E := \begin{pmatrix} R_n(a^{(1)}, \ldots, a^{(n)}) & & & \\ & R_n(b^{(1)}, \ldots, b^{(n)}) & & \\ & & R_n(c^{(1)}, \ldots, c^{(n)}) & \\ & & & \ddots \end{pmatrix}$$

Let $\mathcal{A} = \{A', D_2, \ldots, D_{q-1}, E\}$. Our construction implies that a completion for $\mathcal{A}$ determines a satisfying assignment for ϕ and vice-versa. Thus we see that SIM-COMPLETION(q,q) is NP-complete. ■

4.3 Deciding Non-Singularity

In this section, we obtain a new hardness result for deciding non-singularity of a single mixed matrix where the same indeterminate can appear in multiple entries. This new result is a corollary of Theorem 4.6.

Before doing so, we briefly distinguish two notions of rank for a mixed matrix. The distinction was not important up to this point because the notions are equivalent for mixed matrices where each variable appears only once. The ***term-rank*** of

A is the largest value r such that there exists an $r \times r$ submatrix whose determinant is a non-zero polynomial. The ***generic-rank***[1] of A over $\mathbb{F}_q$ is the largest value r such that there exists a completion of A in $\mathbb{F}_q$ and an $r \times r$ submatrix whose determinant *evaluates* to a non-zero value under this completion. These two notions of rank are not in general the same. In particular, the term-rank does not depend on the field under consideration, but the generic rank does. To see why, consider the matrix $M = \left(\begin{smallmatrix} x & x \\ 1 & x \end{smallmatrix}\right)$. The determinant of this matrix is $x^2 - x = x(x-1)$. This is a non-zero polynomial, but it evaluates to zero at every point in $\mathbb{F}_2$. Thus the term-rank of M is 2 but the generic-rank over $\mathbb{F}_2$ is 1.

We focus on the problem of deciding whether a mixed matrix is non-singular (i.e., has full rank) with respect to its generic-rank. The following hardness result was already known.

Theorem 4.7 (Buss et al. [4]). *Let M be a matrix over any field $\mathbb{F}_q$ where variables can appear any number of times. The problem of deciding whether M is non-singular is NP-complete.*

Our Theorem 4.6 yields as a corollary the following refinement of Theorem 4.7.

Corollary 4.8. *Let M be a matrix over any field $\mathbb{F}_q$ where variables can appear several times. The problem of deciding whether M is non-singular is NP-complete, even if we require that each variable appear in M at most q times.*

Proof. Given a circuit ϕ, construct the set of matrices $\mathcal{A} = \{A', D_2, \ldots, D_{q-1}, E\}$ as in Theorem 4.6. Let M be the block-diagonal matrix with these matrices on the diagonal and note that M satisfies the stated restriction. We have shown that a satisfying assignment for ϕ corresponds to a maximum-rank completion for M and vice-versa. Thus deciding whether M is non-singular with respect to its generic rank also decides whether ϕ is satisfiable. ■

For the sake of comparison, we provide a brief description of the proof of Buss et al. They reduce from 3SAT by arithmetizing the CNF formula. Recall that 3SAT is hard only when each variable appears at least three times. The resulting formula is raised to the power $q-1$ to ensure that its value is either 0 or 1. Next, they use the fact that the determinant function is ***universal***, meaning that for any arithmetic

[1]Our terminology follows that of Murota [36]. Buss et al. [4] use the term ***maxrank*** to refer to the generic-rank.

formula one can construct a (small) matrix whose determinant equals that formula. This construction is originally due to Valiant [47]. The matrix that results from their reduction is hard only when there are at least $3q - 3$ occurrences of some variable. In contrast, Corollary 4.8 shows hardness even when all variables occur at most q times. Thus our result is strictly stronger, even in the case that $q = 2$.

We remark that, given the formula of equation (4.2), Valiant's construction would not yield the matrix R_n. Instead, it would yield a matrix where each variable occurs twice, which is insufficient to prove Theorem 4.6.

Appendix A

Preliminaries on Linear Algebra

In this section we review elementary but useful results from linear algebra. These results may be found in the books of Strang [45], Aitken [2], and Murota [36].

Let us begin with notation. Matrices are denoted with capital letters, such as A. The entry of A in the i^{th} row and j^{th} column is denoted $A_{i,j}$. If I is a subset of the row indices of A and J is a subset of the column indices then $A[I,J]$ denotes the submatrix with rows in I and columns in J. This notation is not terribly convenient for referring to a particular row or column. Instead, we will use the following handy notation. Thinking of $*$ as a "wildcard" character, we will let $A_{i,*}$ denote the i^{th} row of A and let $A_{*,k}$ denote the k^{th} column of A. We occasionally construct submatrices by deleting the i^{th} row and j^{th} column. The notation $A_{\mathrm{del}(i,j)}$ denotes the resulting matrix.

Next, we present a convenient formulation of matrix multiplication. Suppose A and B are matrices of compatible size and that $C = AB$. The standard formula for the entries of C is

$$C_{i,k} = \sum_j A_{i,j} B_{j,k}. \tag{A.1}$$

That is, entry i,j of C is the dot-product of the i^{th} row of A and the k^{th} column of B. Applying our wildcard notation to equation (A.1), we can view matrix multiplication in four convenient ways.

Fact A.1. *The following four reformulations of matrix multiplication are obtained by* *replacing variables in equation* (A.1) *with wildcards in various ways.*

$$\begin{array}{ll}
\textit{Entry-based view:} & C_{i,k} \;=\; A_{i,*} \cdot B_{*,k} \\
\textit{Row-based view:} & C_{i,*} \;=\; \sum_j A_{i,j} B_{j,*} \\
\textit{Column-based view:} & C_{*,k} \;=\; \sum_j A_{*,j} B_{j,k} \\
\textit{Outer-product view:} & C \;=\; C_{*,*} \;=\; \sum_j A_{*,j} B_{j,*}
\end{array}$$

Next, we turn to the determinant. The determinant function and its properties are used extensively in this thesis. We review these properties next.

Fact A.2. *The determinant of an $n \times n$ matrix A is defined as follows.*

$$\det A \;=\; \sum_{\sigma} \operatorname{sign}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)} \tag{A.2}$$

The sum is over all permutations σ of the numbers $\{1, \ldots, n\}$, and $\operatorname{sign}(\sigma)$ *denotes the sign, or signature, of the permutation.*

Fact A.3. *A matrix A has a multiplicative inverse if and only if* $\det A \neq 0$. *Such a matrix is called **non-singular**.*

Fact A.4. $\det(AB) = \det A \cdot \det B$.

Fact A.5. $\det(A^{-1}) = (\det A)^{-1}$.

Fact A.6. *Let A be a matrix and let A' be obtained from A by swapping two adjacent rows or two adjacent columns. Then* $\det A = -\det A'$. *More generally, if A' is obtained from A by reordering the rows and columns arbitrarily, then* $\det A = \pm \det A'$.

Fact A.7. *The determinant is a linear function of every column. More precisely, let A be a square matrix and suppose $A_{*,i} = v + w$ for some column vectors v and w. Construct the matrix $A_{(v)}$ from A by replacing its i^{th} column with the vector v, and construct the matrix $A_{(w)}$ analogously. Then* $\det A = \det A_{(v)} + \det A_{(w)}$.

The quantity $\det A_{\mathrm{del}(i,j)}$ is called the $(i,j)^{\mathrm{th}}$ ***minor*** of A.

Fact A.8. *Let A be a square matrix and let A' be the matrix obtained from A by adding v to the $(i,j)^{th}$ entry. Then* $\det A' = \det A + v \cdot \det A_{\mathrm{del}(i,j)}$.

Fact A.9 (Generalized Laplace expansion). *Let A be an $n \times n$ matrix. Fix a set of rows*

I such that $\emptyset \neq I \subset \{1, \ldots, n\}$*. Then*

$$\det A \;=\; \sum_{\substack{J \subset \{1,\ldots,n\} \\ |J|=|I|}} \det A[I,J] \cdot \det A[\overline{I},\overline{J}] \cdot (-1)^{\sum_{i \in I} i + \sum_{j \in J} j}. \tag{A.3}$$

Proof. See Aitken [2] or Murota [36]. ■

Fact A.10 (Block Determinant). *Let* $A = \left(\begin{smallmatrix} W & X \\ 0 & Y \end{smallmatrix}\right)$ *be a square matrix where* W *and* Y *are square submatrices. Then* $\det A = \det W \cdot \det Y$.

Proof. Let I_Y and J_Y respectively be the rows and columns corresponding to the submatrix Y. Apply Fact A.9 with $I = I_Y$. If $J \neq J_Y$ then $A[I,J]$ contains a zero column and hence $\det A[I,J] = 0$. Thus the only non-zero term in the sum of (A.3) is obtained when $J = J_Y$. ■

We remark that the positions of the submatrices in Fact A.10 are largely irrelevant — reordering the rows and columns affects only the sign of the determinant (Fact A.6). For example, we also have that $\det \left(\begin{smallmatrix} X & W \\ Y & 0 \end{smallmatrix}\right) = \pm \det W \cdot \det Y$, under the assumption that W and Y are square.

Let A be a square matrix of the form $A = \left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$ where Z is square. If Z is non-singular, the matrix $W - XZ^{-1}Y$ is known as the ***Schur complement*** of Z in A. The Schur complement satisfies the following useful property.

Fact A.11.

$$\det \begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \;=\; \det Z \;\cdot\; \det\big(W - XZ^{-1}Y\big).$$

The ***rank*** of a matrix A, denoted $\operatorname{rank} A$, is defined as the largest size of a non-singular submatrix of A. A non-singular matrix is also said to have ***full rank***.

Fact A.12. *If* $A = BC$ *then* $\operatorname{rank} A \leq \min\{\operatorname{rank} B, \operatorname{rank} C\}$.

Our work also relies on properties of the ***Gaussian elimination*** algorithm and its ***pivot*** operations. Consider an $n \times n$ matrix A. A pivot operation simply adds some multiple α_j of a row i to every other row j. Clearly pivoting requires $O(n^2)$ time. A ***column-clearing pivot***, also known as a ***Gauss-Jordan pivot***, chooses the multiples so that, for some column k, all entries except $A_{i,k}$ become zero. When Gaussian elimination uses Gauss-Jordan pivots, it is called Gauss-Jordan elimination.

It is well known that for any non-singular matrix, the Gauss-Jordan algorithm always completes successfully. We will use a slightly more general statement.

Fact A.13. *Let $A = \left(\begin{smallmatrix} W & X \\ 0 & Y \end{smallmatrix}\right)$ be a square matrix where W and Y are square. If A is non-singular, we may perform Gauss-Jordan elimination on the right half of the matrix, pivoting exactly once in each row and column of Y.*

Usually the point of Gauss-Jordan elimination is to turn a matrix A into the *identity matrix*, and in general swapping rows is necessary to achieve this. We will typically be interested in turning A into a matrix where each row and column has only a single non-zero entry. The specific locations and values of these non-zero entries are irrelevant, hence swapping rows is unnecessary to achieve this goal.

Fact A.14. *Suppose we perform Gauss-Jordan elimination on a matrix A. Let the set of rows and columns on which we have pivoted so far be respectively R and C. Then $A[R, C]$ has full rank.*

When performing Gauss-Jordan elimination on a matrix A, pivoting on row i can be viewed as multiplying A on the left by a matrix which equals the identity matrix except in the i^{th} column. In general, a sequence of pivot operations can be described by a matrix S which is the product of these matrices.

Fact A.15. *Suppose we perform a sequence of Gauss-Jordan pivots, described by a matrix S. That is, if P is the pivoted form of A then $P = SA$. If we haven't yet pivoted on row i then the i^{th} column of S is the i^{th} elementary vector.*

Proof. Let R be the set of rows in which we have performed a pivot operation. Then every row has been incremented by some linear combination of the rows in R. Thus,

$$P_{a,*} \;=\; A_{a,*} + \sum_{j \in (R-a)} \alpha_{a,j} A_{j,*},$$

for some coefficients $\alpha_{a,j}$. We can rewrite this equation as

$$P_{a,*} \;=\; \sum_{j} \alpha_{a,j} A_{j,*},$$

where $\alpha_{a,j} = 1$ if $a = j$ and $\alpha_{a,j} = 0$ if $j \neq R$ and $a \neq j$. The row-based view of matrix multiplication (Fact A.1) shows us that these coefficients are precisely the entries of the matrix S. Since $i \notin R$, the entry $S_{a,i}$ is zero if $a \neq i$ and 1 if $a = i$. ■

Appendix B

Preliminaries on Matroids

Matroids are mathematical objects of great importance in combinatorial optimization and with numerous applications in computer science. For example, many of the well-known results concerning spanning trees, bipartite matchings, and graph connectivity are simple consequences of standard theorems in matroid theory.

A comprehensive introduction to matroids and all of their interesting properties would require a whole book. In this thesis, we will require only a few basic definitions and the existence of certain efficient algorithms. A more detailed introduction to matroids can be found in the following books. Cook et al. [5] give an eminently readable introduction to matroids, highlighting the connections to other combinatorial problems. A more detailed treatment can be found in the books of Murota [36] and Schrijver [44]. Oxley's book [39] focuses exclusively on matroid theory, with an emphasis on mathematical properties rather than connections to computer science.

B.1 Definitions

For the reader unfamiliar with matroids, we introduce them via an example.

Example B.1. Suppose one is given the matrix

$$A := \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Our instincts from linear algebra immediately tell us several things about this matrix. First of all, it does not have full rank, since adding the first two columns together produces the third column. However, the first two columns are linearly independent, so they form a ***basis*** for the column-space of A. This tells us that the rank of A is 2. Since bases are quite important, we might wonder if there are other bases for the column-space. Let us denote the columns of A by a, b and c respectively. We find that the following unordered pairs of columns are linearly independent, and hence are bases.

$$\{\ \{a,b\}\,,\ \{a,c\}\,,\ \{b,c\}\ \}$$

We also know instinctively that removing columns from a basis doesn't ruin linear independence. So we can easily enumerate *all* sets of linearly independent columns.

$$\{\ \emptyset,\ \{a\}\,,\ \{b\}\,,\ \{c\}\,,\ \{a,b\}\,,\ \{a,c\}\,,\ \{b,c\}\ \}\,. \tag{B.1}$$

Let's call this collection of sets $\mathcal{I}$, to denote "independent". Now suppose that we forgot what the exact entries of matrix A were. Indeed, this is quite likely since A was defined on the previous page! Nevertheless, we can still deduce various properties about A from the set $\mathcal{I}$. We can still see that A has rank 2 since the set $\{a,b,c\}$ is not a member of $\mathcal{I}$. We can also deduce that column c is not spanned by either column a or column b individually, but it is spanned by column a and b together.

So it seems that much of the structure of A is captured by $\mathcal{I}$. This collection of sets $\mathcal{I}$ is called a ***matroid***, since it is like the matrix but A more abstract: it captures the linear independence properties of A but not the specific numerical entries. ■

Let's explore matroids a bit more formally and write down some familiar properties of linear independence. Formally, a matroid is an ordered pair $\mathrm{M} := (C,\mathcal{I})$, where C is a finite ***ground set*** and $\mathcal{I}$ is a collection of subsets of C. We may think of C as the set of columns of some matrix and $\mathcal{I}$ as the collection of column-sets that are linearly independent. The collection $\mathcal{I}$ must satisfy three basic properties:

1. $\emptyset \in \mathcal{I}$.

2. If $S \in \mathcal{I}$ and $T \subset S$ then $T \in \mathcal{I}$.

3. If $S, T \in \mathcal{I}$ and $|S| > |T|$ then $\exists s \in S$ such that $s \notin T$ and $(T \cup \{s\}) \in \mathcal{I}$.

These properties are all well-known facts from linear algebra. In fact, we used the first two properties above in deriving equation (B.1). We might start to wonder about other properties that $\mathcal{I}$ satisfies. Can we derive additional properties using just the three listed above, or do we need to delve back into our linear algebra toolbox again? Surprisingly, these three properties are enough to derive almost any fact that one could derive using linear algebra. Since these three properties are so important, they are often called the ***matroid axioms***.

Since one can derive many facts from these axioms alone, it no longer seems important that there was a matrix underlying the definition of C and $\mathcal{I}$. Perhaps one could find other set systems that satisfy these axioms? Then one could learn a lot about those set systems by using the consequences of these axioms. It turns out that many familiar mathematical objects do indeed satisfy the matroid axioms: spanning trees, bipartite matchings, etc. Each of these objects leads to a new type of matroid.

We will encounter a few types of matroids in this thesis.

***Linear matroids*:** These are the matroids that can be obtained as the set of linearly independent columns of some matrix. Many of the commonly encountered matroids are in fact linear. More exotic, non-linear matroids do exist, but they will not be discussed herein. For a matrix A, the corresponding matroid is denoted $\mathbf{M}_A$.

***Transversal matroids*:** These are the matroids that are obtained from bipartite matchings. They are discussed further in the following section.

***Free matroids*:** For any ground set C, the free matroid is the pair $(C, \mathcal{I})$ where $\mathcal{I}$ contains *every* subset of C. Free matroids trivially satisfy all three matroid axioms.

***Direct sums*:** This is a means of constructing a new matroid from two given matroids. Suppose $\mathbf{M}_1 = (C_1, \mathcal{I}_1)$ and $\mathbf{M}_2 = (C_2, \mathcal{I}_2)$ are matroids whose ground sets C_1 and C_2 are disjoint. Their direct sum, denoted $\mathbf{M}_1 \oplus \mathbf{M}_2$, is the matroid $(C_1 \cup C_2, \mathcal{I}_3)$ where $\mathcal{I}_3$ contains all sets that are the union of a set in $\mathcal{I}_1$ and a set in $\mathcal{I}_2$. This construction is easy to visualize when $\mathbf{M}_1$ and $\mathbf{M}_2$ are linear matroids, corresponding to matrices A_1 and A_2. Their direct sum is the linear matroid corresponding to the matrix $\left(\begin{smallmatrix} A_1 & 0 \\ 0 & A_2 \end{smallmatrix}\right)$.

We conclude this section with a remark on terminology. Since the purpose of matroids is, in some sense, to abstract away from matrices and linear algebra, the sets in $\mathcal{I}$ are not described as being "linearly independent". Instead, they are simply called ***independent*** sets.

B.2 Bipartite Matchings and Transversal Matroids

Let us now explain how bipartite matchings give rise to a matroid. Let $G = (V^+ \cup V^-, E)$ be a bipartite graph. If M is a matching for G, define $\partial^+ M$ to be the vertices in V^+ covered by M, and define $\partial^- M$ analogously.

Definition B.2. *A set $S \subseteq V^-$ is called **matchable** if there exists a matching M in G such that $S \subseteq \partial^- M$. Let $\mathcal{I}$ be the collection of all matchable subsets of V^-. The pair $(V^-, \mathcal{I})$ is called a **transversal matroid**.*

To show that a transversal matroid is indeed a matroid, we must prove that the three matroid axioms are satisfied. To prove this, we will actually show an important connection between bipartite graphs and matrices. This connection will imply that a transversal matroid can be derived from a matrix and is hence a linear matroid.

Definition B.3. *Let $G = (V^+ \cup V^-, E)$ be a bipartite graph. We define an associated matrix of indeterminates A. The rows of A are indexed by V^+ and the columns are indexed by V^-. For each $(u, v) \in E$, we set $A_{u,v}$ to be a new indeterminate $x_{u,v}$. All other entries of A are zero.*

This construction is clearly invertible: given A, it is easy to construct the graph G. The connection between G and A is important because many combinatorial properties of G translate into algebraic properties of A.

Theorem B.4. *The size of a maximum matching in G is $\operatorname{rank} A$.*

Proof. We prove a slightly simpler result: G has a perfect matching if and only if $\det A \neq 0$. The more general result follows by considering subgraphs of G and submatrices of A.

Let $V^+ = \{u_1, \dots, u_n\}$ and $V^- = \{v_1, \dots, v_n\}$. Consider the familiar permutation expansion of $\det A$.

$$\det A \;=\; \sum_{\sigma} \pm \prod_{i=1}^{n} A_{i,\sigma(i)} \tag{B.2}$$

Suppose that G does not have a perfect matching. This means that for every permutation σ, there is at least one value i such that $(u_i, v_{\sigma(i)}) \notin E$, implying that $A_{i,\sigma(i)} = 0$. Thus every term in Equation (B.2) vanishes, and hence $\det A = 0$.

On the other hand, suppose that G does have a perfect matching. This means that there exists a permutation σ such that for all i we have $(u_i, v_{\sigma(i)}) \in E$, implying that $A_{i,\sigma(i)} = x_{u_i,v_{\sigma(i)}}$. Thus the term in (B.2) corresponding to σ is non-zero. Every non-zero term in (B.2) is a product of a distinct set of variables, and hence there is no cancellation amongst the terms. Thus $\det A \neq 0$. ■

This theorem stems from the classic work of Frobenius [12] and Kőnig [24]. For a detailed history of this work, see Schrijver [43, 44] or Lovász and Plummer [31].

We now return to the discussion of transversal matroids. We use the following fact, which is a consequence of Theorem B.4: for any matching in G, if $U' \subseteq U$ and $V' \subseteq V$ are the vertices covered by the matching then the submatrix $A[U', V']$ is non-singular. Thus, for every matchable subset $S \subseteq V$, there is a square, non-singular submatrix of A whose columns contain S. It follows that the columns of A corresponding to S are linearly independent. Conversely, if a set of columns S in A is linearly independent then there exists a non-singular submatrix with column-set S. Since the determinant of this submatrix is non-zero, there exists a matching covering S. Thus the matchable subsets of V are precisely the sets of independent columns of A. Since the collection of independent column-sets of A is a matroid by definition, the matchable subsets of V also form a matroid. That is, transversal matroids do indeed satisfy the matroid axioms.

B.3 Algorithms

It is often possible to develop efficient algorithms for problems involving matroids by exploiting their structural properties. An oft-cited example is the problem of finding an independent set of maximum weight. Given a matroid $(C, \mathcal{I})$ and a weight-function $w : C \to \mathbb{R}$, one can find in polynomial time a set S in the collection $\mathcal{I}$ that maximizes $\sum_{s \in S} w(s)$. This problem is not considered further in this thesis, but it does provoke some important questions. What does it mean to be "given a matroid"? Since $\mathcal{I}$ can be exponentially larger than C, what does "polynomial time" mean?

In this thesis, matroids are represented using the ***independence oracle model***. This means that a matroid is represented as a pair (C, f_C) where f_C is a function that decides in time polynomial in $|C|$ whether a set $S \subseteq C$ is independent. For a linear matroid constructed from a matrix of numbers, the oracle can test if a set S of columns is linearly independent by executing the Gaussian elimination algorithm and returning true if it completes successfully. For a transversal matroid on a graph $G = (V^+ \cup V^-, E)$, the oracle can test if a set $S \subseteq V^-$ is matchable by computing a maximum matching in the graph induced by $V^+ \cup S$. The oracle returns true if the resulting matching covers S. This maximum matching can be computed in polynomial time by standard algorithms.

One of the fundamental algorithmic problems on matroids is the ***matroid intersection problem***. This problem plays a central role in this thesis. Suppose we have two matroids $\mathbf{M}_1 = (C, \mathcal{I}_1)$ and $\mathbf{M}_2 = (C, \mathcal{I}_2)$ that are defined on the same ground set C. The objective is to find a set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ that maximizes $|S|$. The matroid intersection problem is known to be solvable in polynomial time. For the special (but common) case of linear matroids, Cunningham [8] gives an algorithm requiring only $O(n^3 \log n)$ time.

We now describe two other important problems that are closely related to the matroid intersection problem. The first of these is the ***matroid union problem***. Given $\mathbf{M}_1 = (C, \mathcal{I}_1)$ and $\mathbf{M}_2 = (C, \mathcal{I}_2)$, the objective is to find disjoint sets $S_1 \in \mathcal{I}_1$ and $S_2 \in \mathcal{I}_2$ maximizing $|S_1| + |S_2|$. The second problem is the (bipartite) ***matroid matching problem***. An instance consists of a bipartite graph $G = (V^+ \cup V^-, E)$ and two matroids $\mathbf{M}^+ = (V^+, \mathcal{I}^+)$ and $\mathbf{M}^- = (V^-, \mathcal{I}^-)$. A matching M in G is called an ***independent matching*** if $\partial^+ M \in \mathcal{I}^+$ and $\partial^- M \in \mathcal{I}^-$. The matroid matching problem is to find a maximum cardinality independent matching.

The matroid intersection, matroid union and (bipartite) matroid matching problems are all equivalent: there are simple reductions from any one to any other [44, 36, 13]. Cunningham's algorithm implies that all three of these problems can be solved in $O(n^3 \log n)$ time, for linear matroids. Thus, when faced with a matroid theoretic problem, we can attempt a reduction to whichever of these three problems is more convenient. We remark that the *non*-bipartite matroid matching problem is much more difficult. See, for example, Lovász and Plummer [31].

These three problems generalize naturally by introducing weights. For the intersection and union problems we define a weight function $w : C \to \mathbb{R}$, and for the matching problem we define a weight function $w : E \to \mathbb{R}$. The new objective is to find a solution maximizing the weight of the solution rather than the cardinality. Gabow and Xu [14] developed efficient, but quite complicated, algorithms for these weighted problems on linear matroids. They give an algorithm that requires $O(n^3 \log n \log(nC))$ time, where C is the maximum weight. This algorithm uses only standard matrix operations. Using fast matrix multiplication, they give an (impractical) algorithm with a theoretical running time of $O(n^{2.77} \log C)$.

Bibliography

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] Alexander C. Aitken. *Determinants and Matrices.* Interscience Publishers, New York, ninth edition, 1956.

[3] Mark Berdan. A matrix rank problem. Master's thesis, University of Waterloo, 2003.

[4] Jonathan F. Buss, Gudmund S. Frandsen, and Jeffrey O. Shallit. The computational complexity of some problems in linear algebra. *Journal of Computer and System Sciences*, 58(3):572–596, 1999.

[5] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley, 1997.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[7] Thomas M. Cover and Joy M. Thomas. *Elements of Information Theory*. Wiley, 1991.

[8] William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, November 1986.

[9] Peter Elias, Amiel Feinstein, and Claude E. Shannon. A note on the maximum flow through a network. *IRE Transactions on Information Theory*, IT-2:117–119, 1956.

[10] Meir Feder, Dana Ron, and Ami Tavory. Bounds on linear codes for network multicast. Technical Report ECCC TR03-033, Electronic Colloquium on Computational Complexity, 2003.

[11] Lester R. Ford and Delbert Ray Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[12] Ferdinand G. Frobenius. Über matrizen aus nicht negativen elementen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, pages 456–477, 1912.

[13] Satoru Fujishige. *Submodular Functions and Optimization*, volume 47 of *Annals of Discrete Mathematics*. North-Holland, 1991.

[14] Harold N. Gabow and Ying Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *Journal of Computer and System Sciences*, 53(1):129–147, 1996.

[15] James F. Geelen. Maximum rank matrix completion. *Linear Algebra and its Applications*, 288:211–217, 1999.

[16] James F. Geelen. Matching theory. Lecture notes from the Euler Institute for Discrete Mathematics and its Applications, 2001.

[17] Nicholas J. A. Harvey, David R. Karger, and Kazuo Murota. Deterministic network coding by matrix completion. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 05)*, pages 489–498, 2005.

[18] Nicholas J. A. Harvey, Robert D. Kleinberg, and April Rasala Lehman. On the capacity of information networks. *Special Issue of the IEEE Transactions on Information Theory and the IEEE/ACM Transactions on Networking*. Submitted March 2005.

[19] Tracey Ho, David R. Karger, Muriel Médard, and Ralf Koetter. Network coding from a network flow perspective. In *Proceedings of the IEEE International Symposium on Information Theory*, 2003.

[20] John E. Hopcroft. Personal communication, 2004.

[21] Sidharth Jaggi, Philip A. Chou, and Kamal Jain. Low complexity algebraic multicast network codes. In *Proceedings of the IEEE International Symposium on Information Theory*, page 368, 2003.

[22] Sidharth Jaggi, Peter Sanders, Philip A. Chou, Michelle Effros, Sebastian Egner, Kamal Jain, and Ludo Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*. Submitted July 2003.

[23] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*. To appear.

[24] Dénes Kőnig. Vonalrendszerek és determinánsok [Hungarian; Line systems and determinants]. *Mathematikai és Természettudományi Értesitő*, 33:221–229, 1915.

[25] Monique Laurent. Matrix completion problems. In C.A. Floudas and P.M. Pardalos, editors, *The Encyclopedia of Optimization*, volume III, pages 221–229. Kluwer, 2001.

[26] April Rasala Lehman and Eric Lehman. Complexity classification of network information flow problems. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 04)*, January 2004.

[27] April Rasala Lehman and Eric Lehman. Network coding: Does the model need tuning? In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 05)*, pages 499–504, January 2005.

[28] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufman, 1992.

[29] Shuo-Yen Robert Li, Raymond Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[30] László Lovász. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory, FCT '79*. Akademie-Verlag, Berlin, 1979.

[31] László Lovász and Michael D. Plummer. *Matching Theory*. Akadémiai Kiadó – North Holland, Budapest, 1986.

[32] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[33] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[34] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[35] Kazuo Murota. Mixed matrices – irreducibility and decomposition. In R. A. Brualdi, S. Friedland, and V. Klee, editors, *Combinatorial and Graph-Theoretic Problems in Linear Algebra*, volume 50 of *IMA Volumes in Mathematics and its Applications*, pages 39–71. Springer-Verlag, 1993.

[36] Kazuo Murota. *Matrices and Matroids for Systems Analysis*. Springer-Verlag, 2000.

[37] Kazuo Murota and Masao Iri. Structural solvability of systems of equations — a mathematical formulation for distinguishing accurate and inaccurate numbers in structural analysis of systems. *Japan J. Appl. Math*, 2:247–271, 1985.

[38] Hung Quang Ngo and Ding-Zhu Du. Notes on the complexity of switching networks. In *Switching Networks: Recent Advances*, pages 307–357. Kluwer Academic Publishers, 2001.

[39] James G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[40] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[41] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen. Polynomial time algorithms for network information flow. In *Proceedings of the 15th Annual*

ACM Symposium on Parallel Algorithms and Architectures (SPAA), pages 286–294, 2003.

[42] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 509–517, 2004.

[43] Alexander Schrijver. On the history of combinatorial optimization (till 1960). `http://homepages.cwi.nl/~lex/files/histco.pdf`.

[44] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, 2003.

[45] Gilbert Strang. *Linear Algebra and its Applications*. Thomson Learning, 1988.

[46] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[47] Leslie G. Valiant. Completeness classes in algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computation (STOC)*, pages 249–261, 1979.

[48] Sir Charles Wheatstone. An account of several new instruments and processes for determining the constants of a voltaic circuit. *Philosophical Transactions of the Royal Society of London*, 133:303–329, 1843.