Matchings, Matroids and Submodular Functions

by

Nicholas James Alexander Harvey

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
[June 2008]
May 2008

Author .....
Department of Electrical Engineering and Computer Science
May 15, 2008

Certified by .....
Michel X. Goemans
Professor of Mathematics
Thesis Supervisor

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

Matchings, Matroids and Submodular Functions

by

Nicholas James Alexander Harvey

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis focuses on three fundamental problems in combinatorial optimization: non-bipartite matching, matroid intersection, and submodular function minimization. We develop simple, efficient, randomized algorithms for the first two problems, and prove new lower bounds for the last two problems.

For the matching problem, we give an algorithm for constructing perfect or maximum cardinality matchings in non-bipartite graphs. Our algorithm requires $O(n^\omega)$ time in graphs with n vertices, where $\omega < 2.38$ is the matrix multiplication exponent. This algorithm achieves the best-known running time for dense graphs, and it resolves an open question of Mucha and Sankowski (2004).

For the matroid intersection problem, we give an algorithm for constructing a common base or maximum cardinality independent set for two so-called "linear" matroids. Our algorithm has running time $O(nr^{\omega-1})$ for matroids with n elements and rank r. This is the best-known running time of any linear matroid intersection algorithm.

We also consider lower bounds on the efficiency of matroid intersection algorithms, a question raised by Welsh (1976). Given two matroids of rank r on n elements, it is known that $O(nr^{1.5})$ oracle queries suffice to solve matroid intersection. However, no non-trivial lower bounds are known. We make the first progress on this question. We describe a family of instances for which $(\log_2 3)n - o(n)$ queries are necessary to solve these instances. This gives a constant factor improvement over the trivial lower bound for a certain range of parameters.

Finally, we consider submodular functions, a generalization of matroids. We give three different proofs that $\Omega(n)$ queries are needed to find a minimizer of a submodular function, and prove that $\Omega(n^2/\log n)$ queries are needed to find all minimizers.

Thesis Supervisor: Michel X. Goemans
Title: Professor of Mathematics

Acknowledgments

A thesis is a milestone that makes one reflect on their journey through graduate school. These journeys are not done in isolation. I have traveled with the continual support and encouragement of my wife, Junko. I thank her for weathering the challenges of graduate school with me.

I was tremendously fortunate to have Michel Goemans as my advisor during my Ph.D. program. He is a true role model, both for his insightful research ideas and for his skill at managing an academic life. His constant support has been truly invaluable.

I would also like to acknowledge those who helped to get me here. My parents and sister supported my education for many years. My mathematical background was nurtured early on by my teacher, Craig Newell. When I was getting started in research, Marvin Theimer and John Dunagan provided tremendous support.

MIT is a magical place because of its inhabitants. I thank Madhu Sudan for his helpful advice and avuncular nature. I learned a lot from my friends and coauthors in the theory group, especially April, Bobby, David, Jelani, Kevin, Krzysztof, Mihai, Petar, Sergey, Swastik, and Tasos. The academic side of MIT was counterbalanced by my friends in Eastgate, particularly Ted & Shanshan, Kevin & Elisabeth, and Felix & Celine.

Chapters 2 and 3 of this thesis benefited from many helpful suggestions of Jim Geelen, Michel Goemans, Satoru Iwata, and David Karger. I also thank Vijay Vazirani for correctly suggesting that the matching algorithm in my conference paper could be simplified further.

Chapter 4 arose from innumerable discussions with many people, including Paul Beame, Bill Cunningham, Jim Geelen, Michel Goemans, Gordon James, Laci Lovász, Mihai Pătraşcu, Sasha Postnikov, Ran Raz, Mike Saks, David Woodruff and Sergey Yekhanin. I am greatly indebted for their technical ideas and encouragement.

Chapter 5 is based on joint work and discussions with Michel Goemans, David Karger and Vahab Mirrokni. I thank Lisa Fleischer, Uri Feige and Mohammad Mahdian for valuable suggestions.

Finally, I thank the Natural Sciences and Engineering Research Council of Canada for their financial support.

Contents

Chapter 1

Introduction

This thesis focuses on three fundamental problems in combinatorial optimization: non-bipartite matching, matroid intersection, and submodular function minimization. The computational complexity of these problems has been a topic of much interest for forty years. We contribute to this long line of work by developing simple, efficient, randomized algorithms for the first two problems, and proving new lower bounds for the last two problems.

1.1 Matchings

The ***matching problem*** is: find a largest set of disjoint edges in a given graph. This is a very natural question regarding a very versatile class of mathematical objects. Accordingly, the matching problem has applications in a wide range of areas, including approximation algorithms [13], computer vision [5], personnel assignment [64], scheduling [28], etc.

A priori, it is not at all obvious how to solve the matching problem efficiently. Is the problem computationally tractable? Edmonds [21, 22] recognized this issue, defined a notion of efficient algorithms (the complexity class P), and gave an efficient algorithm for the matching problem. This work played an important role in the development of graph theory, combinatorial optimization, polyhedral combinatorics, and computational complexity theory. The matching theory book [61] gives an extensive treatment of this subject, and uses matchings as a touchstone to develop much of the theory of combinatorial optimization.

Authors	Year	Running Time
Edmonds [22]	1965	$O(n^2m)$
Even and Kariv [27]	1975	$O(\min\{n^{2.5}, \sqrt{n}m \log n\})$
Micali and Vazirani [66]	1980	$O(\sqrt{n}m)$
Rabin and Vazirani [78]	1989	$O(n^{\omega+1})$
Goldberg and Karzanov [37]	2004	$O(\sqrt{n}m \log(n^2/m)/\log n)$
Mucha and Sankowski [69]	2004	$O(n^\omega)$
Sankowski [84]	2005	$O(n^\omega)$
Our algorithm	2006	$O(n^\omega)$

Table 1.1: *A summary of algorithms for the non-bipartite matching problem. The quantities n and m respectively denote the number of vertices and edges in the graph.*

After Edmonds' work, many polynomial time algorithms for the matching problem were developed. Table 1.1 provides a brief summary, and further discussion can be found in [86, §24.4]. As one can see, there was little progress from 1975 until 2004, when Mucha and Sankowski [69] improved the simple algorithm of Rabin and Vazirani [78] from $O(n^{\omega+1})$ time to $O(n^\omega)$ time, where $\omega < 2.38$ is the exponent indicating the time required to multiply two $n \times n$ matrices [16]. A nice exposition of their algorithm is in Mucha's thesis [68].

Unfortunately, most of the algorithms mentioned above are quite complicated; the algorithms of Edmonds and Rabin-Vazirani are perhaps the only exceptions. For example, the Micali-Vazirani algorithm was not formally proven correct until much later [93]. The Mucha-Sankowski algorithm relies on a non-trivial structural decomposition of graphs called the "canonical partition", and uses sophisticated dynamic connectivity data structures to maintain this decomposition online. Mucha writes [68, §6]:

> [The non-bipartite] algorithm is quite complicated and heavily relies on graph-theoretic results and techniques. It would be nice to have a strictly algebraic, and possibly simpler, matching algorithm for general graphs.

Interestingly, for the special case of bipartite graphs, Mucha and Sankowski give a simple algorithm that amounts to performing Gaussian elimination lazily. Unfortunately, this technique seems to break down for general graphs, leading to a

conjecture that there is no $O(n^\omega)$ matching algorithm for non-bipartite graphs that uses only lazy computation techniques [68, §3.4].

This thesis presents a simple, efficient, randomized algorithm for constructing a maximum matching in $O(n^\omega)$ time. The algorithm is based on a lazy updating scheme, and does not require sophisticated data structures or subroutines other than a black-box algorithm for matrix multiplication and inversion. Our work therefore resolves the central open question of Mucha and Sankowski [69], and refutes the conjecture [68] that no such lazy algorithm exists.

Our algorithm, presented in Chapter 2, builds on the Rabin-Vazirani and Mucha-Sankowski algorithms by contributing two ideas. The first is a recursive decomposition of a graph for which every pair of vertices occurs as a base case of that recursion. The second is the observation that, after making numerous localized updates to a matrix, the Sherman-Morrison-Woodbury formula gives a very useful way of making a corresponding update to the inverse of that matrix.

Complete Matlab code for our algorithm (using matrix inversion as a black box) is given in Section 2.8. This can be combined with the Matlab code in Algorithm A.2, which implements fast LU-factorization, and hence inversion, using fast matrix multiplication as a black box. Some fast matrix multiplication algorithms, such as Strassen's [88], can easily be implemented in, say, 25 lines of code. So, altogether, this yields an algorithm solving the matching problem in $O(n^{2.81})$ time and only about 150 lines of Matlab code.

For many graph optimization problems, randomized methods yield the simplest and most efficient solutions. Some examples include the minimum cut [49] and minimum spanning tree [50] problems. This thesis, arguably, adds the non-bipartite matching problem to this list of examples.

1.2 Matroids

Matroids are abstract objects that capture important combinatorial properties of graphs and matrices. Imagine discussing graphs without mentioning vertices, or matrices without mentioning their entries. Then, roughly speaking, you are discussing a matroid.

The two most important optimization problems involving matroids are

- *The greedy algorithm:* find a minimum weight base of a matroid.

- *The matroid intersection problem:* find a common base in two given matroids.

The greedy algorithm is technically an algorithm, not a problem, but it is so fundamental that it justifies conflating the problem with its solution.

To explain these problems, let us draw an analogy to graphs; we defer the formal definition of matroids until Chapter 3. A "base" in a matroid is the analog of a spanning tree in a graph. The matroid greedy algorithm is the analog of Kruskal's algorithm for finding a minimum weight spanning tree.

The matroid intersection problem requires more work to explain. Suppose that we have two graphs $G = (V, E)$ and $H = (V, F)$, together with a bijection $\pi : E \to F$. A common base is a set of edges $T \subseteq E$ such that T that forms a spanning tree for G and $\pi(T)$ forms a spanning tree for H.

This "common spanning tree" problem seems peculiar, but it is actually fairly natural. For example, one can show that the bipartite matching problem is a special case. The more general matroid intersection problem has applications in various areas such as approximation algorithms [11, 36, 43, 52], graph connectivity [31], mixed matrix theory [70], and network coding [42].

The matroid intersection problem can be solved efficiently, as was shown in the pioneering work of Edmonds [23, 24, 26]. This work led to significant developments concerning integral polyhedra [86], submodular functions [30], and convex analysis [71]. Generally speaking, algorithms involving matroids fall into two classes.

- *Oracle algorithms.* These algorithms access the matroid via an ***oracle*** which answers queries about its structure.
- *Linear matroid algorithms.* These algorithms assume that a matroid is given as input to the algorithm as an explicit matrix which represents the matroid.

Both of these models play an important role in this thesis. Linear matroid algorithms only apply to a subclass of matroids known as ***linear matroids***, which we will define in Chapter 3. Most useful matroids arising in applications are indeed linear matroids.

The matroid intersection problem can be solved efficiently in either the linear model or oracle model. Table 1.2 and Table 1.3 provide a brief summary of the existing algorithms. It should be noted that the Gabow-Xu algorithm achieves the running time of $O(nr^{1.62})$ via use of the $O(n^{2.38})$ matrix multiplication algorithm of

Authors	Year	Running Time
Cunningham [19]	1986	$O(nr^2 \log r)$
Gabow and Xu [32, 33]	1989	$O(nr^{1.62})$
Our algorithm	2006	$O(nr^{\omega-1})$

Table 1.2: *A summary of linear matroid algorithms for the matroid intersection problem. The quantities n and r respectively denote the number of columns and rows of the given matrix.*

Authors	Year	Number of Oracle Queries
Edmonds [23][1]	1968	not stated
Aigner and Dowling [1]	1971	$O(nr^2)$
Tomizawa and Iri [90]	1974	not stated
Lawler [55]	1975	$O(nr^2)$
Edmonds [26]	1979	not stated
Cunningham [19]	1986	$O(nr^{1.5})$

Table 1.3: *A summary of oracle algorithms for the matroid intersection problem. The quantities n and r respectively denote the number of elements and rank of the matroid; they are analogous to the quantities n and r mentioned in Table 1.2.*

Coppersmith and Winograd [16]. However, this bound seems somewhat unnatural: for square matrices their running time is $O(n^{2.62})$, although one would hope for a running time of $O(n^{2.38})$.

Chapter 3 of this thesis presents a linear matroid algorithm for the matroid intersection problem that uses only $O(nr^{\omega-1})$ time. The algorithm is randomized and quite simple. We have not implemented it, but it would be straightforward to do so. Whereas most existing matroid algorithms use augmenting path techniques, ours uses an algebraic approach. Several previous matroid algorithms also use algebraic techniques [4, 58, 73]. This approach requires that the given matroids are linear, and additionally requires that the two matroids can be represented as matrices over the same field. These assumptions will be discussed further in Chapter 3.

Is it possible that the algorithms listed at the bottom of Table 1.2 and Table 1.3 are optimal? First let us consider linear matroid algorithms (Table 1.2). If it is

[1] Edmonds [23] gives an efficient algorithm for the matroid partition problem. As was shown by Edmonds [24, 25], this implies an efficient algorithm for the matroid intersection problem.

eventually shown that $\omega = 2$ then our algorithm would require $\tilde{O}(nr)$ time which is near-optimal. But if $\omega > 2$ then proving optimality is hopeless: the best known circuit-size lower bound for *any* explicit function in the complexity class NP is still only linear [45].

Oracle algorithms are a different story. Since these algorithms are formulated in a restricted model of computation (they can only access the matroids via oracle queries), it is conceivable to prove strong lower bounds. To our knowledge, this question was first raised by Welsh[1] [94, p368] in 1976. In the following thirty years, no non-trivial lower bounds were shown for matroid intersection in the oracle model. In this thesis, we prove the first non-trivial result. Chapter 4 describes a family of instances and proves that $(\log_2 3)\, n - o(n)$ queries are necessary to solve these instances. Our proof uses methods from communication complexity and group representation theory.

1.3 Submodular functions

What is a submodular function? One possible answer is: a very general abstraction of a matroid. But that's not quite right; it's like saying that a convex function is an abstraction of a quadratic polynomial. Perhaps a more illuminating answer is: the sensible definition that one obtains when trying to define what it means for a function on $\{0,1\}^n$ to be "convex".

Submodular functions share many familiar properties with convex functions (e.g., one can efficiently find their minimum), and also have some similarities to concave functions (e.g., they reflect some notion of "economies of scale"). They arise in a wide variety of contexts: graph theory, linear algebra, information theory [29], economics [91], etc. In particular, submodular functions play a very important role in combinatorial optimization. The submodular property often leads to elegant, succinct proofs, such as Lovász's proof [57] of Edmonds' disjoint branchings theorem. Submodular functions also often arise as the right-hand vector in large, combinatorially-defined linear programs. Consequently, the problem of separating over such polytopes involves the problem of minimizing a submodular function.

The ***submodular function minimization*** (SFM) problem is: given a submodular

[1] To be precise, Welsh asked about the number of queries needed to solve the matroid partition problem, which is equivalent to matroid intersection, as was mentioned earlier.

function, find a point that minimizes it. This is a very general problem that encompasses many important optimization problems, including matroid intersection and finding a minimum cut in a graph. Accordingly, this problem has received much attention in the literature, dating back to the work of Grötschel, Lovász, and Schrijver [39] in 1981, and even Edmonds [24] in 1970. Given a submodular function, one can find a minimizer (or even find *all* minimizers) using only $O(n^5)$ queries [63] to the function, where n is the dimension of the function's domain.

Several researchers have raised the question of proving a lower bound on the number of queries needed for SFM. (See, for example, the surveys of Iwata [46] or McCormick [63, p387]). In Chapter 5, we prove that n queries are needed; in fact, we give three different proofs that $\Omega(n)$ queries are needed. We also show that $\Omega(n^2/\log n)$ queries are needed to find *all* minimizers.

Finally, we turn to the question of "learning" (or "approximating") a submodular function. How much information can one learn about a submodular function using only $\mathrm{poly}(n)$ queries? Is it possible to approximate the value of f on *every* point in the domain? We show that it is not possible to approximate f better than a factor $\Omega(\sqrt{n/\log n})$.

A subsequent paper by Svitkina and Fleischer [89] has built on our work in very nice ways. They use submodular functions to generalize several classic optimization problems, such as sparsest cut, knapsack, and others. Our result is used to show that any efficient algorithm for "submodular sparsest cut" or "submodular knapsack" must have approximation ratio $\Omega(\sqrt{n/\log n})$. These results are tight: they also show matching upper bounds.

1.4 Preliminaries

We now introduce some mathematical notation and computational assumptions that are used in subsequent chapters.

Basic notation

We will use the following sets of numbers: the integers ($\mathbb{Z}$), the non-negative integers ($\mathbb{N}$), the reals ($\mathbb{R}$), the non-negative reals ($\mathbb{R}_{\geq 0}$), the complex numbers ($\mathbb{C}$), and the finite field on q elements ($\mathbb{F}_q$). The set of integers $\{1, \ldots, n\}$ is denoted $[n]$. The

base-2 logarithm and natural logarithm of x are respectively denoted $\log x$ and $\ln x$.

If X is a set then $X + i$ denotes $X \cup \{i\}$. If X and Y are two sets then $X \oplus Y$ denotes their symmetric difference, i.e., $X \oplus Y = (X \setminus Y) \cup (Y \setminus X)$. The notation $X \dot{\cup} Y$ denotes the union of sets X and Y, and asserts that this is a disjoint union, i.e., $X \cap Y = \emptyset$. The power set (set of all subsets) of a set S is denoted 2^S.

If M is a matrix, a submatrix containing rows S and columns T is denoted $M[S,T]$. A submatrix containing all rows (columns) is denoted $M[*,T]$ ($M[S,*]$). A submatrix $M[S,T]$ is sometimes written as $M_{S,T}$ when this enhances legibility. The i^{th} row (column) of M is denoted $M_{i,*}$ ($M_{*,i}$). An entry of M is denoted $M_{i,j}$. The submatrix obtained by deleting row i and column j (row-set I and column-set J) from M is denoted $M_{\text{del}(i,j)}$ ($M_{\text{del}(I,J)}$).

The notation $\mathbb{R}^E$ denotes the Euclidean space of dimension $|E|$ over $\mathbb{R}$ where the dimensions are indexed by the elements of E. For any subset $U \subseteq E$, its ***characteristic vector*** is the vector $\chi_U \in \{0,1\}^E$ where, for each $e \in E$, $\chi_U(e) = 1$ if $e \in U$ and $\chi_U(e) = 0$ otherwise. For any vector $x \in \mathbb{R}^E$ the notation $x(U)$ denotes $\sum_{e \in U} x(e)$, or equivalently $x(U) = x^{\mathsf{T}} \chi_U$.

Assumptions and conventions

We assume a randomized computational model, in which algorithms have access to a stream of independent, unbiased coin flips. All algorithms presented herein are randomized, even if this is not stated explicitly. Furthermore, our computational model assumes that all arithmetic operations in a given finite field require a single time step, even if we work with an extension field of polynomial size.

A Monte Carlo algorithm is one whose output may be incorrect with some (bounded) probability, but whose running time is not a random variable. The Monte Carlo algorithms that we present have one-sided error and failure probability $\delta < 1/2$. Thus, the error can be decreased to any desired quantity λ by performing $\log \lambda$ independent trials. A Las Vegas algorithm is one whose output is always correct but whose running time is a random variable, typically with bounded expectation.

The value ω is a real number defined as the infimum of all values c such that multiplying two $n \times n$ matrices requires $O(n^c)$ time. We say that matrix multiplication requires $O(n^\omega)$ time although, strictly speaking, this is not accurate. Nevertheless, this inaccuracy justifies the following notational convention: we will im-

plicitly ignore $\text{polylog}(n)$ factors in expressions of the form $O(n^\omega)$.

Linear algebraic algorithms

We conclude this section by considering the algorithmic efficiency of operations on matrices with entries in a field $\mathbb{F}$. We assume that two $n \times n$ matrices can be multiplied in $O(n^\omega)$ time. This same time bound suffices for the following operations.

- *Determinant.* Given an $n \times n$ matrix M, compute $\det M$.
- *Rank.* Given an $n \times n$ matrix M, compute $\text{rank}\, M$.
- *Inversion.* Given a non-singular $n \times n$ matrix M, compute M^{-1}.
- *Max-rank submatrix.* Given an $n \times n$ matrix M, compute sets A and B such that $M[A, B]$ is non-singular and $|A| = |B| = \text{rank}\, M$.

A complete discussion of these algorithms appears in Appendix A.

Consider now the problem of rectangular matrix multiplication. For example, one could multiply an $r \times n$ matrix A by a $n \times r$ matrix B, where $r < n$. This can be accomplished by partitioning A and B into blocks of size $r \times r$, multiplying the i^{th} block of A by the i^{th} block of B via an $O(r^\omega)$ time algorithm, then finally adding these results together. Since $\lceil n/r \rceil$ multiplications are performed, the total time required is $O(nr^{\omega-1})$. This basic technique will frequently be used in the subsequent chapters. More sophisticated rectangular matrix multiplication algorithms [15] do exist, but they will not be considered herein.

Chapter 2

Non-bipartite matching algorithm

In this chapter, we present an efficient, randomized algorithm for constructing perfect or maximum cardinality matchings in non-bipartite graphs. Our algorithm requires $O(n^\omega)$ time in graphs with n vertices. This is the best-known running time for dense graphs, on par with the algorithm of Mucha and Sankowski [69].

The chapter is organized as follows. First, some basic definitions and facts are introduced in Section 2.1. Then Section 2.2 introduces the Tutte matrix, which is used in Section 2.3 to obtain a simple algorithm based on self-reducibility for constructing a perfect matching. Section 2.4 improves the efficiency of this algorithm by using Woodbury updates. Finally, Section 2.5 improves the previous algorithm by introducing a recursive procedure to perform the updates. The resulting algorithm requires only $O(n^\omega)$ time for graphs with n vertices.

Two variants of this algorithm are presented in Section 2.6: the first constructs a maximum cardinality matching, and the second is a Las Vegas algorithm. Some proofs are given in Section 2.7. Section 2.8 presents Matlab code that implements the algorithm of Section 2.5.

2.1 Preliminaries

Let $G = (V, E)$ be a finite, simple, undirected graph with vertex set V and edge set E. A ***matching*** in G is a set of edges $M \subseteq E$ such that every vertex is contained in at most one edge of M. If every vertex is contained in exactly one edge then M is called a ***perfect matching***. Let I and J be disjoint subsets of V. We denote the

edges within set I and edges crossing between I and J as follows.

$$E[I] = \{ \{u,v\} : u \in I, v \in I, \text{ and } \{u,v\} \in E \}$$
$$E[I,J] = \{ \{u,v\} : u \in I, v \in J, \text{ and } \{u,v\} \in E \}$$

The algorithms of this chapter are based on some standard tools from linear algebra, which we review now. Let $\mathbb{F}$ be a field, let $\mathbb{F}[x_1, \ldots, x_m]$ be the ring of polynomials over $\mathbb{F}$ in indeterminates $\{x_1, \ldots, x_m\}$, and let $\mathbb{F}(x_1, \ldots, x_m)$ be the field of rational functions over $\mathbb{F}$ in these indeterminates. A matrix with entries in $\mathbb{F}[x_1, \ldots, x_m]$ or $\mathbb{F}(x_1, \ldots, x_m)$ will be called a ***matrix of indeterminates***. A matrix M of indeterminates is said to be non-singular if its determinant is not the zero function. In this case, M^{-1} exists and it is a matrix whose entries are in $\mathbb{F}(x_1, \ldots, x_m)$. The entries of M^{-1} are given by:

$$(M^{-1})_{i,j} = (-1)^{i+j} \cdot \det M_{\mathrm{del}(j,i)} \,/\, \det M. \tag{2.1}$$

Given a matrix of indeterminates, our algorithms will typically substitute values in $\mathbb{F}$ for the indeterminates. So for much of the discussion below, it suffices to consider ordinary numeric matrices over $\mathbb{F}$.

The following fact, and its corollary, are fundamental to many of our results.

Fact 2.1 (Sherman-Morrison-Woodbury Formula). *Let M be an $n \times n$ matrix, U be an $n \times k$ matrix, and V be a $k \times n$ matrix. Suppose that M is non-singular. Then*

- *$M + UV^{\mathsf{T}}$ is non-singular iff $I + V^{\mathsf{T}}M^{-1}U$ is non-singular*
- *if $M + UV^{\mathsf{T}}$ is non-singular then*

$$(M + UV^{\mathsf{T}})^{-1} = M^{-1} - M^{-1}U\,(I + V^{\mathsf{T}}M^{-1}U)^{-1}\,V^{\mathsf{T}}\,M^{-1}.$$

Proof. This result is well-known; see, e.g., Golub and Van Loan [38, §2.1.3]. We give a proof on page 29. ■

Corollary 2.2. *Let M be a non-singular matrix and let N be its inverse. Let $\tilde{M}$ be a matrix which is identical to M except that $\tilde{M}_{S,S} \neq M_{S,S}$. Then $\tilde{M}$ is non-singular iff*

$$\det\left(I + (\tilde{M}_{S,S} - M_{S,S}) \cdot N_{S,S}\right) \neq 0.$$

If $\tilde{M}$ is non-singular, then

$$\tilde{M}^{-1} = N - N_{*,S}\left(I + (\tilde{M}_{S,S} - M_{S,S})N_{S,S}\right)^{-1}(\tilde{M}_{S,S} - M_{S,S})\,N_{S,*}.$$

The proof of Corollary 2.2 can be found on page 29.

A matrix M is called ***skew-symmetric*** if $M = -M^{\mathsf{T}}$. Note that the diagonal entries of a skew-symmetric matrix are necessarily zero (over a field of characteristic greater than two).

Fact 2.3. *Let M be an $n \times n$ skew-symmetric matrix. If M is non-singular then M^{-1} is also skew-symmetric.*

The proof of Fact 2.3 can be found on page 30.

2.2 Tutte matrix

Let $G = (V, E)$ be a graph with $|V| = n$, and let $\mathcal{M}$ be the set of all perfect matchings of G. A lot of information about $\mathcal{M}$ is contained in the ***Tutte matrix*** T of G, which is defined as follows. For each edge $\{u, v\} \in E$, associate an indeterminate $t_{\{u,v\}}$. Then T is an $n \times n$ matrix where $T_{u,v}$ is $\pm t_{\{u,v\}}$ if $\{u, v\} \in E$ and 0 otherwise. The signs are chosen such that T is skew-symmetric.

We now describe an important polynomial associated with the Tutte matrix. The ***Pfaffian*** of T is defined as

$$\mathrm{Pf}(T) := \sum_{\mu \in \mathcal{M}} \mathrm{sgn}(\mu) \cdot \prod_{\{u,v\} \in \mathcal{M}} T_{u,v},$$

where $\mathrm{sgn}(\mu) \in \{-1, 1\}$ is a sign whose precise definition is not needed for our purposes. Tutte showed several nice properties of T, one of which the following fact.

Fact 2.4 (Tutte [92]). *Consequently, G has a perfect matching iff T is non-singular.*

Proof. This follows from the (previously known) fact that $\det(T) = \mathrm{Pf}(T)^2$. See, e.g., Godsil [35]. ■

This is a useful characterization, but it does not directly imply an efficient algorithm to test if G has a perfect matching. The issue is that $\mathrm{Pf}(T)$ has a monomial

for every perfect matching of G, of which there may be exponentially many. In this case $\det T$ also has exponential size, and so computing it symbolically is inefficient.

Fortunately, Lovász [58] showed that the rank of T is preserved with high probability after randomly substituting non-zero values from a sufficiently large field for the indeterminates. Let us argue the full-rank case more formally. Suppose that G has a perfect matching. Then, over any field, $\mathrm{Pf}(T)$ is a non-zero polynomial of degree $n/2$. Since $\det T = \mathrm{Pf}(T)^2$, $\det T$ is a non-zero polynomial of degree n, again over any field. The Schwartz-Zippel lemma [67, Theorem 7.2] shows that if we evaluate this polynomial at a random point in $\mathbb{F}_q^{|E|}$ (i.e., pick each $t_{\{u,v\}} \in \mathbb{F}_q$ independently and uniformly), then the evaluation is zero with probability at most n/q. Therefore the rank is preserved with probability at least $1 - n/q$.

After this numeric substitution, the rank of the resulting matrix can be computed in $O(n^\omega)$ time, using the algorithm of Appendix A, for example. If the resulting matrix has full rank then G definitely has a perfect matching. Otherwise, we assume that G does not have a perfect matching. This discussion shows that there is an efficient, randomized algorithm to *test* if a graph has a perfect matching (with failure probability at most n/q). The remainder of this chapter considers the problem of *constructing* a perfect matching, if one exists.

2.3 A self-reducibility algorithm

Since Lovász's approach allows one to efficiently test if a graph has a perfect matching, one can use a self-reducibility argument to actually construct a perfect matching. Such an argument was explicitly stated by Rabin and Vazirani [78]. The algorithm deletes as many edges as possible subject to the constraint that the remaining graph has a perfect matching. Thus, at the termination of the algorithm, the remaining edges necessarily form a perfect matching.

The first step is to construct T, then to randomly substitute values for the indeterminates from a field of size q, where q will be chosen below. If T does not have full rank then the algorithm halts and announces that the graph has no perfect matching. Otherwise, it examines the edges of the graph one-by-one. For each edge $\{r, s\}$, we temporarily delete it and test if the resulting graph still has a perfect matching. If so, we permanently delete the edge; if not, we restore the edge.

When temporarily deleting an edge, how do we test if the resulting graph has

a perfect matching? This is done again by Lovász's approach. We simply set $T_{r,s} = T_{s,r} = 0$, then test whether T still has full rank. The Schwartz-Zippel lemma again shows that this test fails with probability at most n/q, even without choosing new random numbers.

Since there are fewer than n^2 edges, a union bound shows that the failure probability is less than n^3/q. If the random values are chosen from a field of size at least n^3/δ, then the overall failure probability is at most δ. The time for each rank computation is $O(n^\omega)$, so the total time required by this algorithm is $O(n^{\omega+2})$. As mentioned earlier, we may set $\delta = 1/2$ and obtain any desired failure probability λ by performing $\log \lambda$ independent trials.

2.4 An algorithm using rank-2 updates

The self-reducibility algorithm can be improved to run in $O(n^4)$ time. To do so, we need an improved method to test if an edge can be deleted while maintaining the property that the graph has a perfect matching. This is done by applying Corollary 2.2 to the matching problem as follows.

Suppose that we have computed the inverse of the Tutte matrix $N := T^{-1}$. Let $\tilde{G}$ denote the graph where edge $\{r, s\}$ has been deleted. We wish to decide if $\tilde{G}$ still has a perfect matching. This can be decided (probabilistically) as follows. Let $\tilde{T}$ be the matrix which is identical to T except that $\tilde{T}_{S,S} = 0$, where $S = \{r, s\}$. We will test if $\tilde{T}$ is non-singular. By Corollary 2.2 and Fact 2.3, this holds if and only if the following determinant is non-zero.

$$\det\left(\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} - \begin{pmatrix}0 & T_{r,s}\\ -T_{r,s} & 0\end{pmatrix}\cdot\begin{pmatrix}0 & N_{r,s}\\ -N_{r,s} & 0\end{pmatrix}\right) = \det\begin{pmatrix}1+T_{r,s}\,N_{r,s} & \\ & 1+T_{r,s}\,N_{r,s}\end{pmatrix}$$

Thus $\tilde{T}$ is non-singular iff $(1 + T_{r,s}\,N_{r,s})^2 \neq 0$. So, to decide if edge $\{r, s\}$ can be deleted, we simply test if $N_{r,s} \neq -1/T_{r,s}$. The probability that this test fails (i.e., if $\tilde{G}$ has a perfect matching but $\tilde{T}$ is singular) is at most n/q, again by the Schwartz-Zippel lemma.

After deleting an edge $\{r, s\}$ the matrix N must be updated accordingly. By

Corollary 2.2, we must set

$$N := N + N_{*,S} \begin{pmatrix} 1 + T_{r,s} N_{r,s} & \\ & 1 + T_{r,s} N_{r,s} \end{pmatrix}^{-1} T_{S,S} N_{S,*}. \tag{2.2}$$

This computation takes only $O(n^2)$ time, since it is a rank-2 update.

The algorithm examines each edge, decides if it can be deleted and, if so, performs the update described above. The main computational work of the algorithm is the updates. There are $O(n^2)$ edges, so the total time required is $O(n^4)$. As in Section 2.3, if the random values are chosen from a field of size at least n^3/δ, then the overall failure probability is at most δ.

2.5 A recursive algorithm

In this section, we describe an improvement of the previous algorithm which requires only $O(n^\omega)$ time. The key idea is to examine the edges of the graph in a particular order, with the purpose of minimizing the cost of updating N. The ordering is based on a recursive partitioning of the graph which arises from the following observation.

Claim 2.5. *Let R and S be subsets of V with $R = R_1 \dot\cup R_2$ and $S = S_1 \dot\cup S_2$. Then*

$$\begin{aligned} E[S] &= E[S_1] \mathbin{\dot\cup} E[S_2] \mathbin{\dot\cup} E[S_1, S_2] \\ E[R, S] &= E[R_1, S_1] \mathbin{\dot\cup} E[R_1, S_2] \mathbin{\dot\cup} E[R_2, S_1] \mathbin{\dot\cup} E[R_2, S_2]. \end{aligned}$$

For the sake of simplicity, let us henceforth assume that n is a power of two. To satisfy this assumption, one may add a new clique on $2^{\lceil \log n \rceil} - n$ vertices that is disconnected from the rest of the graph. One may easily see that the resulting graph has a perfect matching iff the original graph does.

The pseudocode in Algorithm 2.1 examines all edges of the graph by employing the recursive partitioning of Claim 2.5. At each base of the recursion, the algorithm examines a single edge $\{r, s\}$ and decides if it can be deleted, via the same approach as the previous section: by testing if $N_{r,s} \neq -1/T_{r,s}$. As long as we can ensure that $N_{r,s} = (T^{-1})_{r,s}$ in each base of the recursion then the algorithm is correct: the matrix T remains non-singular throughout the algorithm and, at the end, N has exactly

Algorithm 2.1 FINDPERFECTMATCHING *constructs a perfect matching of the graph G, assuming the number of vertices is a power of two. The probability of failure is at most δ if the field $\mathbb{F}$ has cardinality at least $|V|^3/\delta$.* DELETEEDGESWITHIN *deletes any edge $\{r, s\}$ with both $r, s \in S$, subject to the constraint that the graph still has a perfect matching.* DELETEEDGESCROSSING *deletes any edge $\{r, s\}$ with $r \in R$ and $s \in S$, subject to the constraint that the graph still has a perfect matching. Updating N requires $O(|S|^\omega)$ time; details are given below.*

FINDPERFECTMATCHING($G = (V, E)$)
  Let T be the Tutte matrix for G
  Replace the variables in T with random values from field $\mathbb{F}$
  If T is singular, return "no perfect matching"
  Compute $N := T^{-1}$
  DeleteEdgesWithin(V)
  Return the set of remaining edges

DELETEEDGESWITHIN(S)
  If $|S| = 1$ then return
  Partition S into $S_1 \dot\cup S_2$ such that $|S_1| = |S_2| = |S|/2$
  For $i \in \{1, 2\}$
    DeleteEdgesWithin(S_i)
    Update $N[S, S]$
  DeleteEdgesCrossing(S_1, S_2)

DELETEEDGESCROSSING(R, S)
  If $|R| = 1$ then
    Let $r \in R$ and $s \in S$
    If $T_{r,s} \neq 0$ and $N_{r,s} \neq -1/T_{r,s}$ then
      ▷ Edge $\{r, s\}$ can be deleted
      Set $T_{r,s} = T_{s,r} = 0$
      Update $N[R \cup S, R \cup S]$
  Else
    Partition $R = R_1 \dot\cup R_2$ and $S = S_1 \dot\cup S_2$ such that $|R_1| = |R_2| = |S_1| = |S_2| = |R|/2$
    For $i \in \{1, 2\}$ and for $j \in \{1, 2\}$
      DeleteEdgesCrossing(R_i,S_j)
      Update $N[R \cup S, R \cup S]$

one non-zero entry per row and column (with failure probability n^3/δ).

Algorithm 2.1 ensures that $N_{r,s} = (T^{-1})_{r,s}$ in each base case by updating N whenever an edge is deleted. However, the algorithm does not update N all at once, as Eq. (2.2) indicates one should do. Instead, it only updates portions of N that are needed to satisfy the following two invariants.

1. DELETEEDGESWITHIN(S) initially has $N[S,S] = T^{-1}[S,S]$. It restores this property after each recursive call to DELETEEDGESWITHIN(S_i) and after calling DELETEEDGESCROSSING(S_1, S_2).
2. DELETEEDGESCROSSING(R, S) initially has $N[R \cup S, R \cup S] = T^{-1}[R \cup S, R \cup S]$. It restores this property after deleting an edge, and after each recursive call to DELETEEDGESCROSSING(R_i, S_j).

To explain why invariant 1 holds, consider executing DELETEEDGESWITHIN(S). We must consider what happens whenever the Tutte matrix is changed, i.e., whenever an edge is deleted. This can happen when calling DELETEEDGESWITHIN(S_i) or DELETEEDGESCROSSING(S_1, S_2).

First, suppose the algorithm has just recursed on DELETEEDGESWITHIN(S_1). Let T denote the Tutte matrix before recursing and let $\tilde{T}$ denote the Tutte matrix after recursing (i.e., incorporating any edge deletions that occurred during the recursion). Note that T and $\tilde{T}$ differ only in that $\Delta := \tilde{T}[S_1, S_1] - T[S_1, S_1]$ may be non-zero. Since the algorithm ensures that the Tutte matrix is always non-singular, Corollary 2.2 shows that

$$\tilde{T}^{-1} = T^{-1} - (T^{-1})_{*,S_1} \cdot \left(I + \Delta \cdot (T^{-1})_{S_1,S_1}\right)^{-1} \cdot \Delta \cdot (T^{-1})_{S_1,*}.$$

Restricting to the set S, we have

$$(\tilde{T}^{-1})_{S,S} = (T^{-1})_{S,S} - (T^{-1})_{S,S_1} \cdot \left(I + \Delta \cdot (T^{-1})_{S_1,S_1}\right)^{-1} \cdot \Delta \cdot (T^{-1})_{S_1,S}.$$

Let N refer to that matrix's value before recursing. To restore invariant 1, we must compute the following new value for $N[S,S]$.

$$N_{S,S} := N_{S,S} - N_{S,S_1} \cdot \left(I + \Delta \cdot N_{S_1,S_1}\right)^{-1} \cdot \Delta \cdot N_{S_1,S} \tag{2.3}$$

The matrix multiplications and inversions in this computation all involve matrices of size at most $|S| \times |S|$, so $O(|S|^\omega)$ time suffices.

Next, suppose that the algorithm has just called DELETEEDGESCROSSING(S_1, S_2) at the end of DELETEEDGESWITHIN(S). Invariant 2 ensures that

$$N[S,S] \;=\; N[S_1 \cup S_2, S_1 \cup S_2] \;=\; T^{-1}[S_1 \cup S_2, S_1 \cup S_2] \;=\; T^{-1}[S,S]$$

at the end of DELETEEDGESCROSSING(S_1, S_2), and thus invariant 1 holds at the end of DELETEEDGESWITHIN(S).

Similar arguments show how to compute updates such that invariant 2 holds. After deleting an edge $\{r, s\}$, it suffices to perform the following update.

$$\begin{aligned} N_{r,s} &:= N_{r,s} \cdot (1 - T_{r,s}N_{r,s})/(1 + T_{r,s}N_{r,s}) \\ N_{s,r} &:= -N_{r,s} \end{aligned} \tag{2.4}$$

After recursively calling DELETEEDGESCROSSING(R_i, S_j), we perform an update as follows. Let T denote the Tutte matrix before recursing, let $\tilde{T}$ denote the Tutte matrix after recursing, and let $\Delta := (\tilde{T} - T)_{R_i \cup S_j, R_i \cup S_j}$. Then we set

$$N_{R\cup S, R\cup S} := N_{R\cup S, R\cup S} - N_{R\cup S, R_i\cup S_j} \cdot \left(I + \Delta \cdot N_{R_i\cup S_j, R_i\cup S_j}\right)^{-1} \cdot \Delta \cdot N_{R_i\cup S_j, R\cup S}$$

This shows that the algorithm satisfies the stated invariants.

Analysis. Let $f(n)$ and $g(n)$ respectively denote the running time of the functions DELETEEDGESWITHIN(S) and DELETEEDGESCROSSING(R, S), where $n = |R| = |S|$. As argued above, updating N requires only $O(|S|^\omega)$ time, so we have

$$\begin{aligned} f(n) &= 2 \cdot f(n/2) + g(n) + O(n^\omega) \\ g(n) &= 4 \cdot g(n/2) + O(n^\omega). \end{aligned}$$

By a standard analysis of divide-and-conquer recurrence relations [17], the solutions of these recurrences are $g(n) = O(n^\omega)$ and $f(n) = O(n^\omega)$.

As argued in Section 2.4, each test to decide whether an edge can be deleted fails with probability at most n/q, and therefore the overall failure probability is at most n^3/q. Therefore setting $q \geq n^3/\delta$ ensures that the algorithm fails with probability at most δ.

Remarks. Let T and N denote those matrices' values before recursing on DELETEEDGESWITHIN(S_1), and let $\tilde{T}$ and $\tilde{N}$ denote the values after recursing. Invariant 1 (for the subproblem S_1) ensures that $\tilde{N}[S_1, S_1] = \tilde{T}^{-1}[S_1, S_1]$. The update in Eq. (2.3) is performed to ensure that invariant 1 holds for the subproblem S. However, that computation does not require the matrix $\tilde{N}$, only the matrix N — the matrix $\tilde{N}$ is discarded. For this reason, the invariants stated above are somewhat stronger than necessary. In particular, the update in Eq. (2.4) is completely unnecessary, and is omitted from the Matlab code in Section 2.8.

2.6 Extensions

Maximum matching. Algorithm 2.1 is a Monte Carlo algorithm for finding a perfect matching in a non-bipartite graph. If the graph does not have a perfect matching then T is singular and the algorithm reports a failure. An alternative solution would be to find a maximum cardinality matching. This can be done by existing techniques [78, 68], without increasing the asymptotic running time. Let $T_{R,S}$ be a maximum rank square submatrix of T. Then it is known [78] that $T_{S,S}$ is also non-singular, and that a perfect matching for the subgraph induced by S gives a maximum cardinality matching in G.

This suggests the following algorithm. Randomly substitute values for the indeterminates in T from $\mathbb{F}_q$. The submatrix $T_{R,S}$ remains non-singular with probability at least n/q. Find a maximum rank submatrix of T; without loss of generality, it is $T_{R,S}$. This can be done in $O(n^\omega)$ time using the algorithm in Appendix A. Now apply Algorithm 2.1 to $T_{S,S}$ to obtain a matching containing all vertices in S. This matching is a maximum cardinality matching of the original graph.

Las Vegas. The algorithms presented above are Monte Carlo. They can be made Las Vegas by constructing an optimum dual solution — the Edmonds-Gallai decomposition [86, p423]. Karloff [51] showed that this can be done by algebraic techniques, and Cheriyan [12] gave a randomized algorithm using only $O(n^\omega)$ time. If the dual solution agrees with the constructed matching then this certifies that the matching indeed has maximum cardinality. We may choose the field size q so that both the primal algorithm and dual algorithm succeed with constant probability. Thus, the expected number of trials before this occurs is a constant, and hence the algorithm requires time $O(n^\omega)$ time in expectation.

2.7 Proofs

Proof (of Fact 2.1). Note that

$$\begin{pmatrix} I & 0 \\ -U & I \end{pmatrix} \cdot \begin{pmatrix} I & V^{\mathsf{T}} \\ 0 & M + UV^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} I & V^{\mathsf{T}} \\ -U & M \end{pmatrix} = \begin{pmatrix} I & V^{\mathsf{T}}M^{-1} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I + V^{\mathsf{T}}M^{-1}U & 0 \\ -U & M \end{pmatrix}.$$

Taking determinants shows that $\det(M + UV^{\mathsf{T}}) = \det(I + V^{\mathsf{T}}M^{-1}U) \cdot \det(M)$. This proves the first claim. The second claim follows since

$$\begin{aligned} & \Big(M^{-1} - M^{-1}U(I + V^{\mathsf{T}}M^{-1}U)^{-1}V^{\mathsf{T}}M^{-1}\Big) \cdot \Big(M + UV^{\mathsf{T}}\Big) \\ = \; & I + M^{-1}U\Big(I - (I + V^{\mathsf{T}}M^{-1}U)^{-1} - (I + V^{\mathsf{T}}M^{-1}U)^{-1}V^{\mathsf{T}}M^{-1}U\Big)V^{\mathsf{T}} \\ = \; & I + M^{-1}U(I + V^{\mathsf{T}}M^{-1}U)^{-1}\Big((I + V^{\mathsf{T}}M^{-1}U) - I - V^{\mathsf{T}}M^{-1}U\Big)V^{\mathsf{T}} \\ = \; & I, \end{aligned}$$

as required. ■

Proof (of Corollary 2.2). Let us write $\tilde{M} = M + UV^{\mathsf{T}}$, where

$$M = \begin{matrix} \\ S \\ \overline{S} \end{matrix} \begin{matrix} \begin{matrix} S & \overline{S} \end{matrix} \\ \begin{pmatrix} M_{S,S} & M_{S,\overline{S}} \\ M_{\overline{S},S} & M_{\overline{S},\overline{S}} \end{pmatrix} \end{matrix} \qquad \tilde{M} = \begin{matrix} \\ S \\ \overline{S} \end{matrix} \begin{matrix} \begin{matrix} S & \overline{S} \end{matrix} \\ \begin{pmatrix} \tilde{M}_{S,\overline{S}} & M_{S,\overline{S}} \\ M_{\overline{S},S} & M_{\overline{S},\overline{S}} \end{pmatrix} \end{matrix}$$

$$U = \begin{matrix} \\ S \\ \overline{S} \end{matrix} \begin{matrix} S \\ \begin{pmatrix} I \\ 0 \end{pmatrix} \end{matrix} \qquad V^{\mathsf{T}} = \begin{matrix} \\ S \end{matrix} \begin{matrix} \begin{matrix} S & \overline{S} \end{matrix} \\ \begin{pmatrix} \tilde{M}_{S,S} - M_{S,S} & 0 \end{pmatrix} \end{matrix}.$$

Then Fact 2.1 shows that $\tilde{M}$ is non-singular iff $I + V^{\mathsf{T}}M^{-1}U$ is non-singular. Using the definition of N, V and U, we get

$$I + V^{\mathsf{T}}NU \; = \; I + (\tilde{M}_{S,S} - M_{S,S})N_{S,S}.$$

Furthermore, Fact 2.1 also shows that

$$\begin{aligned}\tilde{M}^{-1} &= N - NU\left(I + V^{\mathsf{T}}NU\right)^{-1}V^{\mathsf{T}}N \\ &= N - N_{*,S}\left(I + (\tilde{M}_{S,S} - M_{S,S})N_{S,S}\right)^{-1}(\tilde{M}_{S,S} - M_{S,S})\,N_{S,*},\end{aligned}$$

as required. ■

Proof (of Fact 2.3). Suppose that M^{-1} exists. Then

$$(M^{-1})_{i,j} = \left((M^{-1})^{\mathsf{T}}\right)_{j,i} = \left((M^{\mathsf{T}})^{-1}\right)_{j,i} = \left((-M)^{-1}\right)_{j,i} = -(M^{-1})_{j,i},$$

as required. ■

2.8 Matlab implementation

This section gives Matlab code which implements Algorithm 2.1.

```
%%% FindPerfectMatching %%%
% Input: A is the adjacency matrix of a graph G.
% Output: a vector representing a perfect matching of G.
function match = FindPerfectMatching(A)
global T N

% Build Tutte matrix.
% Add dummy vertices so that size is a power of 2.
% The chosen range of random values ensures success w.p. > 1/2
m=size(A,1); n=pow2(ceil(log2(m))); q=2*n^3;
T=zeros(n);
for i=1:m for j=i+1:m
    if (A(i,j)~=0) T(i,j)=unidrnd(q); T(j,i)=-T(i,j); end;
end; end;
for i=m+1:n for j=i+1:n    % Dummy vertices and edges
    T(i,j)=unidrnd(q); T(j,i)=-T(i,j);
end; end;

if (rank(T)<n)
    match=[]; return;      % No perfect matching
end;
N = inv(T);
DeleteEdgesWithin(1:n);

% T now (probably) has one entry per row and column.
% Return the corresponding matching of G.
[match,z] = find(T);
if (length(match)==n)
    match=match(1:m);
else
    match=[];
end;


%%% DeleteEdgesWithin %%%
function DeleteEdgesWithin(S)
global T N
n=length(S); m=n/2;
if n==1 return; end
for i=1:2
    Si=S(1+(i-1)*m:i*m);
    Told=T(Si,Si); Nold=N(Si,Si);
    DeleteEdgesWithin(Si);
    Delta=T(Si,Si)-Told; N(Si,Si)=Nold;
    N(S,S)=N(S,S)-N(S,Si)*inv(eye(m)+Delta*Nold)*Delta*N(Si,S);
end;
DeleteEdgesCrossing(S(1:m),S(m+1:n));
```

```
%%% DeleteEdgesCrossing %%%
function DeleteEdgesCrossing(R,S)
global T N
n=length(R); m=n/2; RS=[R,S];
if n==1
    r=R(1); s=S(1);
    % Test if T(r,s)<>0 and N(r,s)<>-1/T(r,s), avoiding floating-point error
    if ( abs(T(r,s))>100*eps && abs(N(r,s)+1/T(r,s))>100*eps )
        T(r,s)=0; T(s,r)=0;
    end;
    return;
end
for i=1:2 for j=1:2
    Ri=R(1+(i-1)*m:i*m); Sj=S(1+(j-1)*m:j*m); RiSj=[Ri,Sj];
    Told=T(RiSj,RiSj); Nold=N(RiSj,RiSj);
    DeleteEdgesCrossing(Ri,Sj);
    Delta=T(RiSj,RiSj)-Told; N(RiSj,RiSj)=Nold;
    N(RS,RS)=N(RS,RS)-N(RS,RiSj)*inv(eye(n)+Delta*Nold)*Delta*N(RiSj,RS);
end; end;
```

Chapter 3

Matroid intersection algorithm

In this chapter, we present algorithms for solving the matroid intersection problem for linear matroids that are explicitly represented over the same field. Section 3.1 formally defines matroids and our required properties. Section 3.2 gives an overview of our approach, and Section 3.3 explains the connections to linear algebra. Section 3.4 shows how these tools can be used to give an efficient algorithm for matroids of large rank. That algorithm is then used as a subroutine in the algorithm of Section 3.5, which requires only $O(nr^{\omega-1})$ time for matroids with n elements and rank r.

The algebraic approach used in this chapter only applies when the two given matroids are linear and represented over the same field. The same assumption is needed by several existing algorithms [4, 9, 58, 73]. Although there exist linear matroids for which this assumption cannot be satisfied (e.g., the Fano and non-Fano matroids), this assumption is valid for the vast majority of matroids arising in applications. For example, the regular matroids are those that are representable over all fields; this class includes the graphic, cographic and partition matroids. Many classes of matroids are representable over all but finitely many fields; these include the uniform, matching, and transversal matroids, as well as deltoids and gammoids [86]. Our results apply to any two matroids from the union of these classes.

3.1 Matroids

Matroids are combinatorial objects first introduced by Whitney [95] and others in the 1930s. Many excellent texts contain an introduction to the subject [14, 56, 70, 75, 86, 94]. We review some of the important definitions and facts below.

A ***matroid*** is a combinatorial object defined on a finite ground set S. The cardinality of S is typically denoted by n. There are several important ancillary objects relating to matroids, any one of which can be used to define matroids. Below we list those objects that play a role in this paper, and we use "base families" as the central definition.

Base family: This non-empty family $\mathcal{B} \subseteq 2^S$ satisfies the axiom:

> Let $B_1, B_2 \in \mathcal{B}$. For each $x \in B_1 \setminus B_2$, there exists $y \in B_2 \setminus B_1$ such that $B_1 - x + y \in \mathcal{B}$.

A matroid can be defined as a pair $\mathrm{M} = (S, \mathcal{B})$, where $\mathcal{B}$ is a base family over S. A member of $\mathcal{B}$ is called a ***base***. It follows from the axiom above that all bases are equicardinal. This cardinality is called the ***rank of the matroid*** M, typically denoted by r.

Independent set family: This family $\mathcal{I} \subseteq 2^S$ is defined as

$$\mathcal{I} = \{ I : I \subseteq B \text{ for some } B \in \mathcal{B} \}.$$

A member of $\mathcal{I}$ is called an ***independent set***. Any subset of an independent set is clearly also independent, and a maximum-cardinality independent set is clearly a base.

The independent set family can also be characterized as a non-empty family $\mathcal{I} \subseteq 2^S$ satisfying

- $A \subseteq B$ and $B \in \mathcal{I} \implies A \in \mathcal{I}$;
- $A \in \mathcal{I}$ and $B \in \mathcal{I}$ and $|A| < |B| \implies \exists b \in B \setminus A$ such that $A + b \in \mathcal{I}$.

Rank function: This function, $r : 2^S \to \mathbb{N}$, is defined as

$$r(T) = \max_{I \in \mathcal{I},\, I \subseteq T} |I|.$$

A maximizer of this expression is called a ***base for T*** in M. A set I is independent iff $r(I) = |I|$. Rank functions satisfy the following important inequality, known as the ***submodular inequality***.

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B) \qquad \forall\, A, B \subseteq S \tag{3.1}$$

This inequality is of particular importance, and it leads to the notion of submodular functions, which we discuss further in Chapter 5.

Since all of the objects listed above can be used to characterize matroids, we sometimes write $\mathrm{M} = (S, \mathcal{I})$, or $\mathrm{M} = (S, \mathcal{I}, \mathcal{B})$, etc. To emphasize the matroid associated to one of these objects, we often write $\mathcal{B}_{\mathrm{M}}$, r_{M}, etc.

Examples. Many matroids fall into several useful families.

- *Linear matroids.* Let Q be a matrix over a field $\mathbb{F}$ whose columns are indexed by the set S. Define

$$\mathcal{I} = \{\, I \subseteq S : Q[*, I] \text{ has full column-rank} \,\}.$$

  Then $\mathrm{M} = (S, \mathcal{I})$ is a ***linear matroid*** and Q is called a ***linear representation*** of M. There exist matroids which do not have a linear representation over any field, although many interesting matroids can be represented over some field.

- *Laminar matroids.* Let S be a ground set and let $\mathcal{L} \subset 2^S$ be a laminar family. That is, for $A, B \in \mathcal{L}$, either $A \subseteq B$ or $B \subseteq A$ or $A \cap B = \emptyset$. For each $A \in \mathcal{L}$, associate a value $d_A \in \mathbb{N}$. Define

$$\mathcal{I}_{\mathcal{L},d} = \{\, I \subseteq S : |I \cap A| \leq d_A \; \forall A \in \mathcal{L} \,\}.$$

  Then $(S, \mathcal{I}_{\mathcal{L},d})$ is a ***laminar matroid*** [30, p43].

- *Partition matroids.* Let $S = S_1 \dot\cup S_2 \dot\cup \cdots \dot\cup S_k$ be a partition of the ground set. and let $d_1, \ldots, d_k \in \mathbb{N}$. Define

$$\mathcal{I} = \{\, I \subseteq S : |I \cap S_i| \leq d_i \; \forall i \,\}.$$

Then $\mathrm{M} = (S, \mathcal{I})$ is a ***partition matroid.***

- *Uniform matroids.* Let S be a ground set, let k be an integer, and let

$$\mathcal{I} = \{ I \subseteq S : |I| \leq k \}.$$

Then $\mathrm{M} = (S, \mathcal{B})$ is the ***uniform matroid*** of rank k on S.

One may easily verify that uniform matroids are a special case of partition matroids, which are a special case of a laminar matroids, which are a special case of linear matroids.

One important operation on matroids is ***contraction.*** Let $\mathrm{M} = (S, \mathcal{B})$ be a matroid. Given a set $T \subseteq S$, the contraction of M by T, denoted M/T, is defined as follows. Its ground set is $S \setminus T$. Next, fix a base B_T for T in M, so $B_T \subseteq T$ and $r_{\mathrm{M}}(T) = r_{\mathrm{M}}(B_T)$. The base family of M/T is defined as:

$$\mathcal{B}_{\mathrm{M}/T} = \{ B \subseteq S \setminus T : B \cup B_T \in \mathcal{B}_{\mathrm{M}} \}.$$

Thus, for $I \subseteq S \setminus T$, we have

$$I \in \mathcal{I}_{\mathrm{M}/T} \iff I \cup B_T \in \mathcal{I}_{\mathrm{M}}. \tag{3.2}$$

The rank function of M/T satisfies:

$$r_{\mathrm{M}/T}(X) = r_{\mathrm{M}}(X \cup T) - r_{\mathrm{M}}(T). \tag{3.3}$$

Suppose two matroids $\mathrm{M}_1 = (S, \mathcal{B}_1)$ and $\mathrm{M}_2 = (S, \mathcal{B}_2)$ are given. A set $B \subseteq S$ is called a ***common base*** if $B \in \mathcal{B}_1 \cap \mathcal{B}_2$. A ***common independent set*** (or an ***intersection***) is a set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$. The ***matroid intersection problem*** is to construct a common base of M_1 and M_2. The decision version of the problem is to decide whether a common base exists. The optimization version of the problem is to construct an intersection of M_1 and M_2 with maximum cardinality. Edmonds [24] proved the following important min-max relation which gives a succinct certificate of correctness for the matroid intersection problem.

Fact 3.1 (Matroid Intersection Theorem). *Let* $\mathrm{M}_1 = (S, \mathcal{I}_1, r_1)$ *and* $\mathrm{M}_2 = (S, \mathcal{I}_2, r_2)$

be given. Then

$$\max_{I \in \mathcal{I}_1 \cap \mathcal{I}_2} |I| \;=\; \min_{A \subseteq S} \big(r_1(A) + r_2(S \setminus A) \big).$$

Matroid Models. To specify a matroid requires, in general, space that is exponential in the size of the ground set since there are $2^{2^{O(n)}}$ paving matroids — see Section 4.3. With such an enormous representation, many matroid problems have trivial algorithmic solutions whose running time is polynomial in the input length.

This observation motivates two different models for representing matroids, as discussed in Chapter 1.

- *Linear model.* Instead of representing all matroids, this model restricts to the class of linear matroids. This is a reasonable assumption since most of the matroids that arise in practice are actually linear. Such matroids can be represented by a matrix giving a linear representation of the matroid, as described above. There always exists such a matrix of size at most $r \times n$.
- *Oracle model.* In this model, a matroid $\mathrm{M} = (S, \mathcal{I})$ is represented by an ***oracle*** — a subroutine which answers queries about the structure of M. The most common such oracle is an ***independence oracle***, which answers the following queries: given a set $A \subseteq S$, is $A \in \mathcal{I}$?

This chapter focuses exclusively on the linear model.

3.2 Overview of algorithm

We now give a high-level overview of the algorithms. First, some notation and terminology are needed. Let $\mathrm{M}_1 = (S, \mathcal{B}_1)$ and $\mathrm{M}_2 = (S, \mathcal{B}_2)$. Our algorithms will typically assume that M_1 and M_2 have a common base; the goal is to construct one. Any subset of a common base is called an ***extensible*** set. If J is extensible, $i \in S \setminus J$, and $J + i$ is also extensible then i is called ***allowed*** (relative to J).

The general idea of our algorithm is to build a common base incrementally. For example, suppose that $\{b_1, \ldots, b_r\}$ is an arbitrary common base. Then

- $\emptyset$ is extensible,
- b_1 is allowed relative to $\emptyset$,
- b_2 is allowed relative to $\{b_1\}$,

Algorithm 3.1 *A general overview of our algorithm for constructing a common base of two matroids* $\mathbf{M}_1$ *and* $\mathbf{M}_2$.

MATROIDINTERSECTION($\mathbf{M}_1$, $\mathbf{M}_2$)
  Set $J = \emptyset$
  For each $i \in S$, do
    Invariant: J is extensible
    Test if i is allowed (relative to J)
    If so, set $J := J + i$

- b_3 is allowed relative to $\{b_1, b_2\}$, etc.

So building a common base is straightforward, so long as we can test whether an element is allowed, relative to the current set J. This strategy is illustrated in Algorithm 3.1. The following section provides linear algebraic tools that we will later use to test whether an element is allowed.

3.3 Formulation using linear algebra

Suppose that each $\mathbf{M}_i = (S, \mathcal{B}_i)$ is a linear matroid representable over a common field $\mathbb{F}$. Let $r_i : S \to \mathbb{N}$ be the rank function of $\mathbf{M}_i$. Let Q_1 be an $r \times n$ matrix whose *columns* represent $\mathbf{M}_1$ over $\mathbb{F}$ and let Q_2 be a $n \times r$ matrix whose *rows* represent $\mathbf{M}_2$ over $\mathbb{F}$. For notational convenience, we will let Q_1^J denote $Q_1[*, J]$ and Q_2^J denote $Q_2[J, *]$. So $J \in \mathcal{I}_2$ iff Q_2^J has full row-rank.

Let T be a diagonal matrix whose rows and columns are indexed by the set S where entry $T_{i,i}$ is an indeterminate t_i. Define

$$Z := \begin{pmatrix} 0 & Q_1 \\ Q_2 & T \end{pmatrix}.$$

For $J \subseteq S$, let $T(J)$ denote $T_{\mathrm{del}(J,J)}$, the matrix obtained by deleting the rows and columns in J. For each $J \subseteq S$, we also define the matrix

$$Z(J) := \begin{pmatrix} 0 & Q_1^J & Q_1^{\overline{J}} \\ Q_2^J & 0 & 0 \\ Q_2^{\overline{J}} & 0 & T(J) \end{pmatrix}. \tag{3.4}$$

So Z and $Z(\emptyset)$ are identical. Let $\lambda(J)$ denote the maximum cardinality of an intersection in the contracted matroids $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$, which were defined in Section 3.1.

The following theorem, whose proof is on page 49, gives a connection between the matrix $Z(J)$ and the quantity $\lambda(J)$.

Theorem 3.2. *For any $J \subseteq S$, we have* $\operatorname{rank} Z(J) = n + r_1(J) + r_2(J) - |J| + \lambda(J)$.

For the special case $J = \emptyset$, this result was stated by Geelen [34] and follows from the connection between matroid intersection and the Cauchy-Binet formula, as noted by Tomizawa and Iri [90]. Building on Theorem 3.2, we obtain the following result which forms the foundation of our algorithms. Its proof is on page 50. Let us now assume that both $\mathbf{M}_1$ and $\mathbf{M}_2$ have rank r, i.e., $r = r_1(S) = r_2(S)$.

Theorem 3.3. *Suppose that $\lambda(\emptyset) = r$, i.e., $\mathbf{M}_1$ and $\mathbf{M}_2$ have a common base. For any $J \subseteq S$ (not necessarily an intersection), $Z(J)$ is non-singular iff J is extensible.*

The preceding theorems lead to the following lemma which characterizes allowed elements. Here, we identify the elements of $S \setminus J$ with the rows and columns of the submatrix of $T(J)$ in $Z(J)$.

Lemma 3.4. *Suppose that $J \subseteq S$ is an extensible set and that $i \in S \setminus J$. The element i is allowed iff $(Z(J)^{-1})_{i,i} \neq t_i^{-1}$.*

Proof. By Theorem 3.3, our hypotheses imply that $Z(J)$ is non-singular. Then element i is allowed iff $Z(J+i)$ is non-singular, again by Theorem 3.3. Note that $Z(J+i)$ is identical to $Z(J)$ except that $Z(J+i)_{i,i} = 0$. Corollary 2.2 implies that $Z(J+i)$ is non-singular iff

$$\det\left(1 - Z(J)_{i,i} \cdot \left(Z(J)^{-1}\right)_{i,i}\right) \neq 0.$$

Eq. (3.4) shows that $Z(J)_{i,i} = t_i$, so the proof is complete. ■

The structure of the matrix Z, and its inverse, will play a key role in our algorithms below. To describe Z^{-1}, let us introduce some properties of Schur complements, proofs of which are given on page 51.

Fact 3.5 (Schur Complement). *Let M be a square matrix of the form*

$$M = \begin{array}{c} \\ S_1 \\ S_2 \end{array} \begin{array}{c} \begin{array}{cc} S_1 & S_2 \end{array} \\ \begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \end{array}$$

where Z is square. If Z is non-singular, the matrix $C := W - XZ^{-1}Y$ is known as the ***Schur complement*** *of Z in M. The Schur complement satisfies many useful properties, two of which are:*

- $\det M = \det Z \cdot \det C$.
- *Let $C_{A,B}$ be a maximum rank square submatrix of C. Then $M_{A \cup S_2, B \cup S_2}$ is a maximum rank square submatrix of M.*

Let Y denote the Schur complement of T in Z, i.e., $Y = -Q_1 \cdot T^{-1} \cdot Q_2$. One may verify (by multiplying with Z) that

$$Z^{-1} = \begin{pmatrix} Y^{-1} & -Y^{-1} \cdot Q_1 \cdot T^{-1} \\ -T^{-1} \cdot Q_2 \cdot Y^{-1} & T^{-1} + T^{-1} \cdot Q_2 \cdot Y^{-1} \cdot Q_1 \cdot T^{-1} \end{pmatrix}. \tag{3.5}$$

Our algorithms cannot directly work with the matrix $Z(J)$ since its entries contain indeterminates. A similar issue was encountered in Chapter 2: for example, $\det Z(J)$ is a polynomial which may have exponential size. This issue is again resolved through randomization. Suppose that $Z(J)$ is non-singular over $\mathbb{F}$, i.e., $\det Z(J)$ is a non-zero polynomial with coefficients in $\mathbb{F}$. Suppose that $\mathbb{F} = \mathbb{F}_{p^c}$ is finite and let $c' \geq c$. Evaluate $\det Z(J)$ at a random point over the extension field $\mathbb{F}_{p^{c'}}$ by picking each $t_i \in \mathbb{F}_{p^{c'}}$ uniformly at random. This evaluation is zero with probability at most n/q, where $q = p^{c'}$, as shown by the Schwartz-Zippel lemma [67]. This probability can be made arbitrarily small by choosing q as large as desired. If $\mathbb{F}$ is infinite then we simply need to choose each t_i uniformly at random from a subset of $\mathbb{F}$ of size q.

3.4 An algorithm for matroids of large rank

This section presents an algorithm which behaves as follows. It is given two matrices Q_1 and Q_2 over $\mathbb{F}$ representing matroids M_1 and M_2, as in the previous section.

The algorithm will decide whether the two matroids have a common base and, if so, construct one. The algorithm requires time $O(n^\omega)$, and is intended for the case $n = O(r)$, i.e., the rank is large.

For the sake of simplicity, let us henceforth assume that n and r are both powers of two. To satisfy this assumption, we replace Q_1 with the matrix

$$\begin{array}{c} \begin{array}{ccc} S & S' & S'' \end{array} \\ \begin{pmatrix} Q_1 & 0 & 0 \\ 0 & I & 0 \end{pmatrix} \end{array}$$

where $|S'| := 2^{\lceil \log r \rceil} - r$ and $|S''| := 2^{\lceil \log |S \cup S'| \rceil} - |S \cup S'|$. In matroid terminology, the elements in S' are coloops and the elements in S'' are loops. An analogous construction is applied to Q_2. It is clear that a common base of the resulting matroids yields a common base of the original matroids.

The algorithm maintains an extensible set J, initially empty, and computes $Z(J)^{-1}$ to help decide which elements are allowed. As elements are added to J, the matrix $Z(J)^{-1}$ must be updated accordingly. A recursive scheme is used to do this, as in the matching algorithm of Chapter 2. Pseudocode is shown in Algorithm 3.2.

First let us argue the correctness of the algorithm. The base cases of the algorithm examine each element of the ground set in order. For each element i, the algorithm decides whether i is allowed relative to J using Lemma 3.4; if so, i is added to J. Thus the behavior of Algorithm 3.2 is identical to Algorithm 3.1, and its correctness follows.

The algorithm decides whether i is allowed by testing whether $\left(Z(J)^{-1}\right)_{i,i} \neq t_i$. (Note that invariant 2 ensures $N_{i,i} = \left(Z(J)^{-1}\right)_{i,i}$.) Lemma 3.4 shows that this test is correct when the t_i's are indeterminates. When the t_i's are random numbers, the probability that this test fails (i.e., i is allowed but $\left(Z(J)^{-1}\right)_{i,i} = t_i$) is at most n/q, again by the Schwartz-Zippel lemma. By a union bound over all elements, the probability of failure is at most δ so long as $q \geq n^2/\delta$.

We now complete the description of the algorithm by explaining how to compute the matrix $M = \left(Z(J \cup J_1)^{-1}\right)_{S_2,S_2}$ during the recursive step. First, note that $N_{S_2,S_2} = \left(Z(J)^{-1}\right)_{S_2,S_2}$. Next, note that $Z(J \cup J_1)$ is identical to $Z(J)$ except that

Algorithm 3.2 *A recursive algorithm to compute a common base of two matroids* $\mathbf{M}_1 = (S, \mathcal{B}_1)$ *and* $\mathbf{M}_2 = (S, \mathcal{B}_2)$ *when* $n = |S| = O(r)$. *We assume that* n *and* r *are both powers of two.*

MATROIDINTERSECTION($\mathbf{M}_1$,$\mathbf{M}_2$)
  Let S be the ground set of $\mathbf{M}_1$ and $\mathbf{M}_2$
  Construct Z and assign random values to the indeterminates $t_1, \ldots, t_n$
  Compute $N := (Z^{-1})_{S,S}$
  J = BUILDINTERSECTION(S, $\emptyset$, N)
  Return J

BUILDINTERSECTION(S, J, N)
  Invariant 1: J is an extensible set
  Invariant 2: $N = \left(Z(J)^{-1}\right)_{S,S}$
  If $|S| \geq 2$ then
    Partition $S = S_1 \dot{\cup} S_2$ where $|S_1| = |S_2| = |S|/2$
    J_1 = BUILDINTERSECTION(S_1, J, N_{S_1,S_1})
    Compute $M := \left(Z(J \cup J_1)^{-1}\right)_{S_2,S_2}$, as described below
    J_2 = BUILDINTERSECTION(S_2, $J \cup J_1$, M)
    Return $J_1 \cup J_2$
  Else
    This is a base case: S consists of a single element $i = a = b$
    If $N_{i,i} \neq t_i^{-1}$ (i.e., element i is allowed) then
      Return $\{i\}$
    Else
      Return $\emptyset$

$Z(J \cup J_1)_{J_1,J_1} = 0$. It follows from Corollary 2.2 that

$$Z(J \cup J_1)^{-1} = Z(J)^{-1} + \left(Z(J)^{-1}\right)_{*,J_1} \left(I - Z(J)_{J_1,J_1} \left(Z(J)^{-1}\right)_{J_1,J_1}\right)^{-1} Z(J)_{J_1,J_1} \left(Z(J)^{-1}\right)_{J_1,*}.$$

Thus

$$\left(Z(J \cup J_1)^{-1}\right)_{S_2,S_2} = N_{S_2,S_2} + N_{S_2,J_1} \left(I - Z_{J_1,J_1} N_{J_1,J_1}\right)^{-1} Z_{J_1,J_1} N_{J_1,S_2}.$$

The matrix M is computed according to this equation, which requires time at most $O(|S|^\omega)$ since all matrices have size at most $|S| \times |S|$.

We now argue that this algorithm requires $O(n^\omega)$ time. The work is dominated by the matrix computations. Computing the initial matrix Z^{-1} clearly takes $O(n^\omega)$

Algorithm 3.3 *The algorithm to compute a common base of two matroids* $\mathbf{M}_1 = (S, \mathcal{B}_1)$ *and* $\mathbf{M}_2 = (S, \mathcal{B}_2)$. *We assume that* n *and* r *are both powers of two.*

MATROIDINTERSECTION($\mathbf{M}_1$,$\mathbf{M}_2$)
  Construct Z and assign random values to the indeterminates $t_1, \ldots, t_n$
  Compute $Y := -Q_1 T^{-1} Q_2$ (used below for computing N)
  Partition $S = S_1 \cup \cdots \cup S_{n/r}$, where $|S_i| = r$
  Set $J := \emptyset$
  For $i = 1$ to n/r do
    Compute $N := (Z(J)^{-1})_{S_i,S_i}$
    J' = BUILDINTERSECTION(S_i,J,N)
    Set $J := J \cup J'$
  Return J

time since Z has size $(n+r) \times (n+r)$. As shown above, computing M requires time $O(|S|^\omega)$. Thus the running time of BUILDINTERSECTION is given by the recurrence

$$f(n) \;=\; 2 \cdot f(n/2) + O(n^\omega),$$

which has solution $f(n) = O(n^\omega)$.

3.5 An algorithm for matroids of any rank

This section builds upon the algorithm of the previous section and obtains an algorithm with improved running time when $r = o(n)$. As before, we assume than n and r are both powers of two. The high-level idea is as follows: partition the ground set S into parts of size r, then execute the BUILDINTERSECTION subroutine from Algorithm 3.2 on each of those parts. For each part, executing BUILDINTERSECTION requires $O(r^\omega)$ time. Since there are n/r parts, the total time required is $O(nr^{\omega-1})$. More detailed pseudocode is given in Algorithm 3.3.

Algorithm 3.3 is correct for the same reasons that Algorithm 3.2 is: each element i is examined exactly once, and the algorithm decides if i is allowed relative to the current set J. Indeed, all decisions made by Algorithm 3.3 are performed in the BUILDINTERSECTION subroutine, which was analyzed in the previous section. Correctness follows immediately, and again the failure probability is δ so long as $q \geq n^2/\delta$.

Let us now analyze the time required by Algorithm 3.3. First, let us consider

the matrix Y, which is computed in order to later compute the matrix N. Since Q_1 has size $r \times n$ and T is diagonal of size $n \times n$, their product $Q_1 T^{-1}$ can be computed in $O(nr)$ time. Since $Q_1 T^{-1}$ has size $r \times n$ and Q_2 has size $n \times r$, their product can be computed in time $O(nr^{\omega-1})$.

Now let us consider the time for each loop iteration. Each call to BUILDINTERSECTION requires $O(r^\omega)$ time, as argued in the previous section. The following claim shows that computing the matrix N also requires $O(r^\omega)$ time. Thus, the total time required by all loop iterations is $(n/r) \cdot O(r^\omega) = O(nr^{\omega-1})$, and therefore Algorithm 3.3 requires $O(nr^{\omega-1})$ time in total.

Claim 3.6. *In each loop iteration, the matrix N can be computed in $O(r^\omega)$ time.*

To prove this, we need another claim.

Claim 3.7. *Suppose that the matrix Y has already been computed. For any $A, B \subseteq S$ with $|A| \leq r$ and $|B| \leq r$, the submatrix $(Z^{-1})_{A,B}$ can be computed in $O(r^\omega)$ time.*

Proof. As shown in Eq. (3.5), we have

$$(Z^{-1})_{S,S} = T^{-1} + T^{-1} Q_2 Y Q_1 T^{-1}.$$

Thus,

$$(Z^{-1})_{A,B} = T^{-1}_{A,B} + (T^{-1}Q_2)_{A,*} Y^{-1} (Q_1 T^{-1})_{*,B}.$$

The submatrices $(T^{-1}Q_2)_{A,*}$ and $(Q_1 T^{-1})_{*,B}$ can be computed in $O(r^2)$ time since T is diagonal. The remaining matrices have size at most $r \times r$, so all computations require at most $O(r^\omega)$ time. ∎

Proof (of Claim 3.6). Note that $Z(J)$ is identical to Z except that $Z(J)_{J,J} = 0$. It follows from Corollary 2.2 that

$$Z(J)^{-1} = Z^{-1} + (Z^{-1})_{*,J} \left(I - Z_{J,J} (Z^{-1})_{J,J}\right)^{-1} Z_{J,J} (Z^{-1})_{J,*}.$$

Thus,

$$\left(Z(J)^{-1}\right)_{S_i,S_i} = (Z^{-1})_{S_i,S_i} + (Z^{-1})_{S_i,J} \left(I - Z_{J,J} (Z^{-1})_{J,J}\right)^{-1} Z_{J,J} (Z^{-1})_{J,S_i}. \quad (3.6)$$

By Claim 3.7, the submatrices $(Z^{-1})_{S_i,S_i}$, $(Z^{-1})_{S_i,J}$, $(Z^{-1})_{J,J}$, and $(Z^{-1})_{J,S_i}$ can all be

computed in $O(r^\omega)$ time. Thus the matrix $N = \left(Z(J)^{-1}\right)_{S_i,S_i}$ can be computed in $O(r^\omega)$ time, as shown in Eq. (3.6). ■

3.6 Extensions

3.6.1 Maximum cardinality intersection

The algorithms presented in the previous sections construct a common base of the two given matroids, if one exists. If the matroids do not have a common base, then the matrix Z is singular and the algorithm reports a failure. An alternative solution would be to find a maximum cardinality intersection rather than a common base. Algorithm 3.3 can be adapted for this purpose, while retaining the running time of $O(nr^{\omega-1})$. We will use the same approach used in Chapter 2 to get a maximum matching algorithm: restrict attention to a maximum-rank submatrix of Z.

Suppose the two given matroids $\mathrm{M}_1 = (S, \mathcal{B}_1)$ and $\mathrm{M}_2 = (S, \mathcal{B}_2)$ do not have a common base. By Theorem 3.2, $\operatorname{rank} Z = n+\lambda$, where λ is the maximum cardinality of an intersection of the two given matroids. Since T is non-singular, Fact 3.5 shows that there exists a row-set A and column-set B, both disjoint from S, such that $|A| = |B| = \operatorname{rank} Z - n$ and $Z_{A\cup S,B\cup S}$ is non-singular. The matrix $Z_{A\cup S,B\cup S}$ has the following form.

$$Z_{A\cup S,\,B\cup S} \;=\; \begin{array}{c} \\ A \\ S \end{array} \begin{array}{c} \begin{array}{cc} B & S \end{array} \\ \begin{pmatrix} & Q_1[A,*] \\ Q_2[*,B] & T \end{pmatrix} \end{array}$$

Now Algorithm 3.3 can be used to find a common base J for the matroids M_A corresponding to $Q_1[A,*]$ and M_B corresponding to $Q_2[*,B]$. Then $Q_1[A,J]$ has full column-rank, which certainly implies that $Q_1[*,J]$ does too, so $J \in \mathcal{I}_1$. Similarly, $J \in \mathcal{I}_2$. Since

$$|J| \;=\; |A| \;=\; \operatorname{rank} Z - n \;=\; \lambda,$$

then $|J|$ is a maximum cardinality intersection for M_1 and M_2.

To analyze the time required by this algorithm, it suffices to focus on the time required to construct the sets A and B. Let $Y = -Q_1TQ_2$ be the Schur complement of T in Z. By Fact 3.5, if $Y_{A,B}$ is a maximum-rank square submatrix of Y then A

and B give the desired sets. As remarked earlier, Y can be computed in $O(nr^{\omega-1})$ time, and a maximum-rank square submatrix can be found in $O(r^{\omega})$ time, via the algorithm of Appendix A. This shows that a maximum cardinality intersection can be constructed in $O(nr^{\omega-1})$ time.

3.6.2 A Las Vegas algorithm

The algorithms presented above are Monte Carlo. In this section, we show that they can be made Las Vegas by constructing an optimal dual solution, i.e., a minimizing set A in Fact 3.1. This dual solution can also be constructed in $O(nr^{\omega-1})$ time. If an optimal dual solution is constructed then this certifies that the output of the algorithm is correct. Since this event occurs with constant probability, the expected number of trials before this event occurs is only a constant.

To construct a dual solution, we turn to the classical combinatorial algorithms for matroid intersection, such as Lawler's algorithm [56]. Expositions can also be found in Cook et al. [14] and Schrijver [86]. Given an intersection, this algorithm searches for ***augmenting paths*** in an ***auxiliary graph.*** If an augmenting path is found, then the algorithm constructs a larger intersection. If no such path exists, then the intersection has maximum cardinality and an optimal dual solution can be constructed.

The first step is to construct a (purportedly maximum) intersection J using the algorithm of Section 3.6.1. We then construct the auxiliary graph for J and search for an augmenting path. If one is found, then J is not optimal, due to the algorithm's unfortunate random choices; this happens with only constant probability. Otherwise, if there is no augmenting path, then we obtain an optimal dual solution. It remains to show that we can construct the auxiliary graph and search for an augmenting path in $O(nr^{\omega-1})$ time.

The auxiliary graph is defined as follows. We have two matroids $\mathrm{M}_1 = (S, \mathcal{I}_1)$ and $\mathrm{M}_2 = (S, \mathcal{I}_2)$, and an intersection $J \in \mathcal{I}_1 \cap \mathcal{I}_2$. The auxiliary graph is a directed, bipartite graph $G = (V, A)$ with bipartition $V = J \dot{\cup} (S \setminus J)$. The arcs are $A = A_1 \dot{\cup} A_2$ where

$$A_1 := \{ (x, y) : y \in J,\ x \notin J,\ \text{and}\ J - y + x \in \mathcal{I}_1 \}$$
$$A_2 := \{ (y, x) : y \in J,\ x \notin J,\ \text{and}\ J - y + x \in \mathcal{I}_2 \}.$$

There are two distinguished subsets $X_1, X_2 \subseteq S \setminus J$, defined as follows.

$$X_1 := \{ x : x \notin J \text{ and } J + x \in \mathcal{I}_1 \}$$
$$X_2 := \{ x : x \notin J \text{ and } J + x \in \mathcal{I}_2 \}$$

It is possible that $X_1 \cap X_2 \neq \emptyset$. Any minimum-length path from X_1 to X_2 is an augmenting path. So J is a maximum cardinality intersection iff G has no directed path from X_1 to X_2. When this holds, the set U of vertices which have a directed path to X_2 satisfies $|J| = r_1(U) + r_2(S \setminus U)$, so U is an optimum dual solution. Schrijver [86, p706] gives proofs of these statements.

Since the auxiliary graph has only n vertices and $O(nr)$ arcs, we can search for a path from X_1 to X_2 in $O(nr)$ time.

Claim 3.8. *The auxiliary graph can be constructed in $O(nr^{\omega-1})$ time.*

Proof. Since $J \in \mathcal{I}_1$, the submatrix $Q_1[*, J]$ has full column-rank. Let $Q_1[I, J]$ be a non-singular square submatrix, and assume for notational simplicity that $I = J = \{1, \ldots, |J|\}$. So Q_1 can be decomposed as follows.

$$Q_1 = \begin{array}{c} \\ I \\ \overline{I} \end{array} \begin{array}{c} \begin{array}{cc} J & \overline{J} \end{array} \\ \begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \end{array}$$

For any non-singular matrix M, the matrix $M \cdot Q_1$ is also a linear representation of M_1. We choose

$$M = \begin{pmatrix} W^{-1} & 0 \\ -YW^{-1} & I \end{pmatrix},$$

so that

$$M \cdot Q_1 = \begin{pmatrix} I & W^{-1}X \\ 0 & Z - YW^{-1}X \end{pmatrix}.$$

Note that $C := Z - YW^{-1}X$ is the Schur complement of $Q_1[I, J]$ in Q_1. It follows from Fact 3.5 that $Q_1[*, J+i]$ has full column-rank iff $C_{*,i}$ contains a non-zero entry. Thus we will add i to X_1 iff $C_{*,i}$ contains a non-zero entry.

We now have the following scenario.

$$M \cdot Q_1 = \begin{array}{c} \\ I \\ \overline{I} \end{array} \begin{array}{c} \begin{array}{ccc} J & X_1 & \overline{J \cup X_1} \end{array} \\ \begin{pmatrix} I & W^{-1}X_{*,X_1} & W^{-1}X_{*,\overline{X_1}} \\ 0 & C_{*,X_1} & 0 \end{pmatrix} \end{array} \tag{3.7}$$

Clearly $Q_1[*, J-i+j]$ has full column-rank iff $(M \cdot Q_1)[*, J-i+j]$ has full-column rank. For $j \in \overline{J \cup X_1}$, it is clear from Eq. (3.7) that the latter condition holds iff $(M \cdot Q_1)_{i,j} \neq 0$.

The arcs in A_1 are constructed as follows. For any $x \in X_1$, we have $J + x \in \mathcal{I}_1$, which implies that $J + x - y \in \mathcal{I}_1 \; \forall y \in J$. Thus $(x, y) \in A_1$ for all $x \in X_1$ and $y \in J$. For any $x \in S \setminus (J \cup X_1)$, we have $(x, y) \in A_1$ iff $(M \cdot Q_1)_{i,j} \neq 0$, as argued above.

The computational cost of this procedure is dominated by the time to compute $M \cdot Q_1$, which clearly takes $O(nr^{\omega-1})$ time. A symmetric argument shows how to build X_2 and A_2. ■

3.7 Discussion

The material of this chapter raises questions for future study.

- Can fast rectangular matrix multiplication [15] be used to obtain more efficient matroid intersection algorithms?
- Can the algorithms of this chapter and Chapter 2 be combined to solve the ***matroid matching*** problem [56, 61, 86] in $O(n^\omega)$ time?
- Are there more efficient algorithms for special matroids, e.g., the intersection of two graphic matroids?

3.8 Proofs

Proof of Theorem 3.2

The following result is useful for analyzing matrices of indeterminates.

Lemma 3.9 (Murota [70, p139]). *Let Q and X be matrices with row-set and column-set $\mathcal{S}$. Suppose that the entries of Q are numbers in some field, and the entries of X contain either a distinct indeterminate or zero. Then*

$$\operatorname{rank}(Q+X) = \max_{A_r \subseteq \mathcal{S},\, A_c \subseteq \mathcal{S}} \big(\operatorname{rank} Q[A_r, A_c] + \operatorname{rank} X[\mathcal{S} \setminus A_r,\, \mathcal{S} \setminus A_c] \big). \tag{3.8}$$

Proof (of Theorem 3.2). The approach is to apply Lemma 3.9 to the matrix $Z(J)$. We have

$$\begin{array}{c} \begin{array}{c} \\ R \\ J \\ S \setminus J \end{array} \begin{array}{c} \begin{array}{ccc} R & J & S \setminus J \end{array} \\ \begin{pmatrix} & Q_1^J & Q_1^{\overline{J}} \\ Q_2^J & & \\ Q_2^{\overline{J}} & & T(J) \end{pmatrix} \end{array} \\ \underbrace{\qquad\qquad\qquad}_{Z(J)} \end{array}$$

$$= \underbrace{\begin{array}{c} \begin{array}{c} \\ R \\ J \\ S \setminus J \end{array} \begin{array}{c} \begin{array}{ccc} R & J & S \setminus J \end{array} \\ \begin{pmatrix} & Q_1^J & Q_1^{\overline{J}} \\ Q_2^J & & \\ Q_2^{\overline{J}} & & \end{pmatrix} \end{array} \end{array}}_{Q} + \underbrace{\begin{array}{c} \begin{array}{c} \\ R \\ J \\ S \setminus J \end{array} \begin{array}{c} \begin{array}{ccc} R & J & S \setminus J \end{array} \\ \begin{pmatrix} & & \\ & & \\ & & T(J) \end{pmatrix} \end{array} \end{array}}_{X}$$

where $\mathcal{S} = R \dot\cup S$ indexes the rows and columns of Z.

Our proof successively adds constraints to the sets A_r and A_c in Eq. (3.8) without changing the maximum value. First, we add the constraint $R \cup J \subseteq A_r$ because those rows clearly cannot contribute to $\operatorname{rank} X[\mathcal{S} \setminus A_r, \mathcal{S} \setminus A_c]$. A similar argument holds for A_c. Next, if $i \in A_r \setminus A_c$ then column i cannot contribute to $\operatorname{rank} X[\mathcal{S} \setminus A_r, \mathcal{S} \setminus A_c]$, since T (and hence X) are diagonal. The same argument applies to $A_c \setminus A_r$, so we may assume without loss of generality that $A_r = A_c$. For notational simplicity, we now drop the subscripts and simply write $A = R \cup A'$, where $A' \subseteq S$.

Consider the $\operatorname{rank} Q[A, A]$ term of Eq. (3.8). Observe that Q_1 and Q_2 occupy disjoint rows and columns of Z, so $\operatorname{rank} Q[A, A] = r_1(A') + r_2(A')$. The second term is $\operatorname{rank} X[\mathcal{S} \setminus A, \mathcal{S} \setminus A] = |\mathcal{S} \setminus A| = |S \setminus A'| = n - |A'|$. Thus we have

$$\operatorname{rank} Z(J) \;=\; \max_{A' \subseteq S} \big(r_1(A') + r_2(A') + n - |A'| \big).$$

Recall that $J \subseteq A'$. Thus we may write $A' = I \dot{\cup} J$, where $I \cap J = \emptyset$. Using the definition of $r_{\mathbf{M}_1/J}$ in Eq. (3.3), we obtain

$$\operatorname{rank} Z(J) \;=\; \max_{I \subseteq S \setminus J} \big(r_{\mathbf{M}_1/J}(I) + r_{\mathbf{M}_2/J}(I) - |I| \big) + n + r_1(J) + r_2(J) - |J|. \quad (3.9)$$

We now argue that this expression is maximized when I is a maximum cardinality intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$, so that $\lambda(J) = |I| = r_{\mathbf{M}_i/J}(I)$ for each i. This shows that $\operatorname{rank} Z(J) \geq \lambda(J) + n + r_1(J) + r_2(J) - |J|$.

To complete the proof, we show the reverse inequality. To do so, let I be a maximizer of Eq. (3.9) that additionally minimizes $2|I| - r_{\mathbf{M}_1/J}(I) - r_{\mathbf{M}_2/J}(I)$. Suppose that this latter quantity is not zero. Then for some i we have $r_{\mathbf{M}_i/J}(I) < |I|$, so there exists $a \in I$ with $r_{\mathbf{M}_i/J}(I - a) = r_{\mathbf{M}_i/J}(I)$. It follows that the set $I - a$ is also a maximizer of Eq. (3.9), contradicting our choice of I. Hence $r_{\mathbf{M}_i/J}(I) = |I|$ for both i. Thus I is an intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$ satisfying $\operatorname{rank} Z(J) = |I| + n + r_1(J) + r_2(J) - |J|$. Since $|I| \leq \lambda(J)$, the desired inequality holds. ■

Proof of Theorem 3.3

Claim 3.10. *Let J be an intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$. J is extensible iff $\lambda(\emptyset) = \lambda(J) + |J|$.*

Proof. Suppose that J is an extensible intersection. This means that there exists an intersection I with $J \subseteq I$ and $|I| = \lambda(\emptyset)$. Since $J \in \mathcal{I}_i$, it is a base for itself in $\mathbf{M}_i$. Thus, Eq. (3.2) shows that $I \setminus J \in \mathcal{I}_{\mathbf{M}_i/J}$ for both i, implying that $\lambda(J) \geq |I \setminus J| = \lambda(\emptyset) - |J|$.

To show the reverse inequality, let I be a maximum cardinality intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$. So $|I| = \lambda(J)$. Then $I \cup J$ is an intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$, showing that $\lambda(\emptyset) \geq \lambda(J) + |J|$. This establishes the forward direction.

Now suppose that $\lambda(J) = \lambda(\emptyset) - |J|$. Then there exists an intersection I of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$ with $|I| = \lambda(\emptyset) - |J|$. Then $I \cup J$ is an intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$, of cardinality $|I| + |J| = \lambda(\emptyset)$. This shows that J is contained in a maximum cardinality intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$, and therefore is extensible. ■

Claim 3.11. *Assume that $\mathbf{M}_1$ and $\mathbf{M}_2$ have the same rank r. For any set $J \subseteq S$, we have $\lambda(J) \leq r - \max_{i \in \{1,2\}} r_i(J)$.*

Proof. Note that $\lambda(J)$ is at most the rank of $\mathbf{M}_i/J$, which is $r - r_i(J)$. ■

Suppose that J is an extensible intersection. This implies that $r_1(J) = |J|$ and $r_2(J) = |J|$ and $\lambda(J) = \lambda(\emptyset) - |J|$, by Claim 3.10. Theorem 3.2 then shows that

$$\begin{aligned} \operatorname{rank} Z(J) &= n + r_1(J) + r_2(J) - |J| + \lambda(J) \\ &= n + |J| + |J| - |J| + \lambda(\emptyset) - |J| \\ &= n + r, \end{aligned}$$

and hence $Z(J)$ is non-singular as required.

We now argue the converse. Clearly $r_2(J) \leq |J|$ and, by Claim 3.11, we have $\lambda(J) + r_1(J) \leq r$. Thus

$$\begin{aligned} \operatorname{rank} Z(J) &= n + r_1(J) + r_2(J) - |J| + \lambda(J) \\ &= n + \big(r_1(J) + \lambda(J)\big) + \big(r_2(J) - |J|\big) \\ &\leq n + r. \end{aligned}$$

If $Z(J)$ is non-singular then equality holds, so we have $r_2(J) = |J|$ and $r_1(J) + \lambda(J) = r$. By symmetry, we also have $r_1(J) = |J|$, implying that J is an intersection. Altogether this shows that $|J| + \lambda(J) = r$, implying that J is also extensible.

Proof of Fact 3.5

Note that

$$\begin{pmatrix} W - XZ^{-1}Y & 0 \\ 0 & Z \end{pmatrix} = \underbrace{\begin{pmatrix} I & -XZ^{-1} \\ 0 & I \end{pmatrix}}_{N_1} \cdot \begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \cdot \underbrace{\begin{pmatrix} I & 0 \\ -Z^{-1}Y & I \end{pmatrix}}_{N_2}. \tag{3.10}$$

Taking the determinant of both sides gives

$$\det\big(W - XZ^{-1}Y\big) \cdot \det Z = \det M,$$

since $\det N_1 = \det N_2 = 1$. This proves the first property.

Furthermore, since N_1 and N_2 have full rank, we have

$$\operatorname{rank} M = \operatorname{rank}\big(W - XZ^{-1}Y\big) + \operatorname{rank} Z = \operatorname{rank} C + |S_2|. \tag{3.11}$$

Now suppose that $C_{A,B}$ is non-singular with $|A| = |B| = \operatorname{rank} C$. Then we have

$$\begin{pmatrix} C_{A,B} & 0 \\ 0 & Z \end{pmatrix} = \begin{pmatrix} I & -X_{A,*}Z^{-1} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} W_{A,B} & X_{A,*} \\ Y_{*,B} & Z \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ -Z^{-1}Y_{*,B} & I \end{pmatrix}. \quad (3.12)$$

It follows from Eq. (3.11) and Eq. (3.12) that

$$\operatorname{rank} M = \operatorname{rank} C + \operatorname{rank} Z = \operatorname{rank} C_{A,B} + \operatorname{rank} Z = \operatorname{rank} M_{A \cup S_2, B \cup S_2},$$

which proves the second property.

Chapter 4

Query lower bounds for matroid intersection

This chapter considers the number of queries needed to solve matroid intersection in the independence oracle model. To be more specific, we consider the decision version of the problem: do two given matroids have a common base?

Let us review the known upper bounds. As stated earlier, the best result is due to Cunningham [19]. He gives a matroid intersection algorithm using only $O(nr^{1.5})$ independence oracle queries for matroids of rank r. It would be truly remarkable if one could show that this is optimal. (For example, it might suggest that the Hopcroft-Karp algorithm [44] for bipartite matching is "morally" optimal.) Unfortunately, we are very far from being able to show anything like that: even a super-linear lower bound is not presently known.

How could one prove a super-linear lower bound on the number of queries needed to solve matroid intersection? This would require that $r = \omega(1)$, since Cunningham's algorithm implies a bound of $O(n)$ for any constant r. One can use dual matroids to show that $n - r = \omega(1)$ is also necessary to obtain a super-linear lower bound. So the rank cannot be too large or too small. Since one can adjust the rank by padding arguments (for example, see Section 4.1.3 below), it suffices to prove a super-linear lower bound for $r = n/2$.

This chapter describes three lower bounds on the number of queries needed, as illustrated in Figure 4-1. Two of these are elementary: we show in Section 4.1 that $2n - 2$ queries are needed for matroids of rank 1, and n queries are needed for matroids of rank $n - 1$. In Section 4.2, we use more involved techniques to show

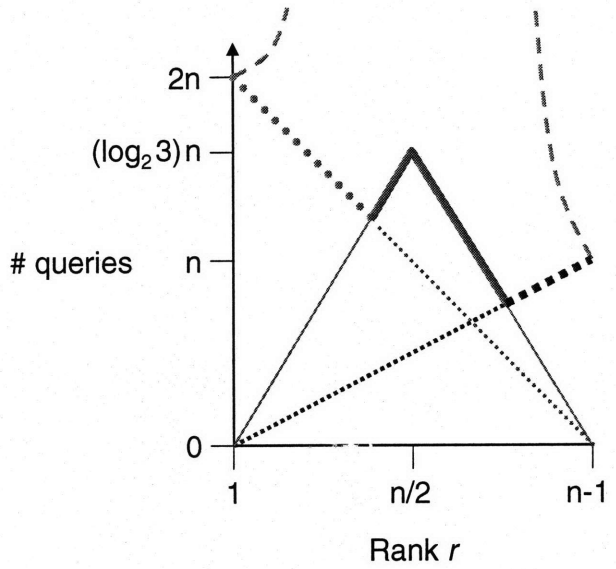


Figure 4-1: This chart reflects our knowledge concerning the number of independence oracle queries needed to solve matroid intersection for matroids with ground set size n and rank r. The purple, dashed lines (which are not to scale) correspond to Cunningham's upper bound of $O(nr^{1.5})$ queries, and a "dual" algorithm which is more efficient for matroids of large rank. The remaining lines correspond to lower bounds, proven in the following sections: Section 4.1.1 (red, round dots), Section 4.1.2 (blue, square dots), and Section 4.2 (green, solid). The best lower bound, corresponding to the upper envelope of these lines, is indicated with thick lines.

that $(\log_2 3) \cdot n - o(n)$ queries are necessary when $r = n/2$. This is, to our knowledge, the only non-trivial progress on Welsh's question from 1976, which we paraphrase as: what is a lower bound on the number of oracle queries needed to solve matroid intersection?

4.1 Elementary lower bounds

4.1.1 Adversary argument for rank-1 matroids

We begin with some easy observations using matroids of rank one. Let S be a finite ground set with $|S| = n$. Let $\emptyset \neq X \subseteq S$ be arbitrary, and let $\mathcal{B}(X) = \{ \{x\} : x \in X \}$. It is easy to verify that $\mathcal{B}(X)$ is the family of bases of a rank one

matroid, which we denote $\mathbf{M}(X)$. Let $\mathcal{M} = \{ \mathbf{M}(X) : \emptyset \neq X \subseteq S \}$. Given two sets $S_0, S_1 \subseteq S$, the two matroids $\mathbf{M}(S_0)$ and $\mathbf{M}(S_1)$ have a common base iff $S_0 \cap S_1 \neq \emptyset$.

We show the following simple theorem.

Theorem 4.1. *Any deterministic algorithm that performs fewer than $2n - 2$ queries cannot solve the matroid intersection problem when given two matroids in $\mathcal{M}$.*

We will prove this theorem in a rather pedantic manner, since the following section requires a similar proof for a slightly less obvious result. Let us first introduce some terminology. Let $Y_i \subseteq S$ be the set of "yes" elements y for which we have decided $\{y\} \in \mathcal{B}(S_i)$. Similarly, let $N_i \subseteq S$ be the set of "no" elements y for which we have decided $\{y\} \notin \mathcal{B}(S_i)$. Let us define the following predicates concerning the adversary's responses to the queries.

Consistent: $\forall\, i \in \{0,1\}\,,\ Y_i \cap N_i = \emptyset$

No-Extensible: $Y_0 \cap Y_1 = \emptyset$

Yes-Extensible: $N_0 \cup N_1 \neq S$

Intuitively, the responses are Consistent if they are valid responses corresponding to some matroid. They are No-Extensible if there exist matroids $\mathbf{M}(S_0)$ and $\mathbf{M}(S_1)$ that do not have a common base and are consistent with the query responses given so far. Yes-Extensible is analogous.

Proof. If $n = 1$ there is nothing to prove, so assume $n \geq 2$. To prove the theorem, we will describe an adversary which replies to the queries of the algorithm and ensures that the responses are Consistent, No-Extensible and Yes-Extensible. The adversary initially adds distinct elements to Y_0 and Y_1, thereby ensuring that $|Y_0| = |Y_1| = 1$ and hence the two matroids do not have rank 0. Let q denote the number of queries performed so far. The adversary maintains two additional properties:

Property 1: $|Y_0 \cup Y_1| + |N_0 \cup N_1| \leq q + 2$

Property 2: $N_i \subseteq Y_{1-i}$

The adversary behaves roughly as follows. The first time a singleton set $\{a\}$ is queried, it returns Yes. Whenever $\{a\}$ is subsequently queried in the other matroid, it returns No. A more formal description is given in the following pseudocode.

Algorithm 4.1 Adversarial responses to the independence oracle queries. The adversary decides whether $A \in \mathcal{I}_i$.

QUERY(i, A)
  If $|A| = 0$, return Yes
  If $|A| > 1$, return No
  Let a be the unique element in A
  If $a \in Y_{1-i}$, add a to N_i and return No
  Add a to Y_i and return Yes

Let us check the correctness of this adversary. First of all, the empty set is independent in every matroid so if $|A| = 0$ then the adversary must return Yes. The adversary is behaving as a rank one matroid, so every independent set has size at most one. So if $|A| > 1$ then the adversary must return No.

So let us suppose that $A = \{a\}$ and $a \in Y_{1-i}$. The No-Extensible property implies $a \notin Y_i$. So adding a to N_i does not violate the Consistent property. Both Y_0 and Y_1 are unchanged so the No-Extensible property is preserved. The algorithm adds a only to N_i so property 1 is preserved. Since $a \in Y_{1-i}$, property 2 is preserved. We now claim that the Yes-Extensible property is maintained, so long as $q < 2n-2$. Combining property 1 and 2, we get

$$2 \cdot |N_0 \cup N_1| \leq |Y_0 \cup Y_1| + |N_0 \cup N_1| \leq q+2 \quad \Longrightarrow \quad |N_0 \cup N_1| \leq (q+2)/2 < n.$$

Thus $N_0 \cup N_1 \neq S$, so the responses are Yes-Extensible.

Similar arguments establish correctness for the case $a \notin Y_{1-i}$. Since the adversary's responses are both No-Extensible and Yes-Extensible, the algorithm cannot have decided whether the two matroids have a common base. ■

The lower bound presented above is essentially tight.

Claim 4.2. *There exists a deterministic algorithm using only $2n$ queries that decides the matroid intersection problem for matroids in $\mathcal{M}$.*

Proof. For every $s \in S$, decide whether $\{s\} \in \mathcal{B}(S_1)$ and $\{s\} \in \mathcal{B}(S_2)$. This takes $2n$ queries, and the algorithm completely learns the set S_1 and S_2. Deciding whether they are disjoint is now trivial. ■

4.1.2 Adversary argument for large-rank matroids

For any $\emptyset \neq X \subseteq S$, let $\mathcal{B}^*(X) = \{ S - x : x \in X \}$, let $\mathrm{M}^*(X) = (S, \mathcal{B}^*(X))$, and let $\mathcal{M}^* = \{ \mathrm{M}^*(X) : \emptyset \neq X \subseteq S \}$. (In matroid terminology, $\mathrm{M}^*(X)$ is the dual matroid for $\mathrm{M}(X)$.) The matroids in $\mathcal{M}^*$ all have rank $n-1$. As above, $\mathrm{M}^*(S_0)$ and $\mathrm{M}^*(S_1)$ have a common base iff $S_0 \cap S_1 \neq \emptyset$. These matroids satisfy the following useful property.

Claim 4.3. *Let $Z \subseteq S$. Then $S \setminus Z$ is an independent set in $\mathrm{M}^*(X)$ iff $X \cap Z \neq \emptyset$.*

Proof. Suppose that $z \in X \cap Z$, so $S - z \in \mathcal{B}^*(X)$. Then $S \setminus Z$ is independent, since $S - Z \subseteq S - z$. Conversely, suppose that $S \setminus Z$ is independent. Then there exists some set $S - z \in \mathcal{B}^*(X)$ with $S \setminus Z \subseteq S - z$. Thus $z \in X$ and $z \in Z$, as required. ■

Theorem 4.4. *Let $n = |S| \geq 2$. Any deterministic algorithm that performs fewer than n queries cannot solve the matroid intersection problem when given two matroids in $\mathcal{M}^*$.*

As above, let $Y_i \subseteq S$ be the set of elements y for which we have decided that $S - y \in \mathcal{B}^*(S_i)$. And let $N_i \subseteq S$ be the set of elements y for which we have decided that $S - y \notin \mathcal{B}^*(S_i)$. The predicates are again:

Consistent: $\forall\, i \in \{0, 1\}\,,\ Y_i \cap N_i = \emptyset$

No-Extensible: $Y_0 \cap Y_1 = \emptyset$

Yes-Extensible: $N_0 \cup N_1 \neq S$

Proof. Let $q < n$ be the number of queries performed so far. The adversary also maintains two properties:

Property 1: $|Y_0 \cup Y_1| \leq q$

Property 2: $N_i \subseteq Y_{1-i}$

The adversary's behavior is described in the following pseudocode.

Algorithm 4.2 Adversarial responses to the independence oracle queries.

Query(i, A): Decide if $S \setminus A \in \mathcal{I}_i$
  If $A \cap Y_i \neq \emptyset$, return Yes
  If $A \not\subseteq Y_{1-i}$
    Pick $a \in A \setminus Y_{1-i}$, and add a to Y_i
    Return Yes
  Set $N_i \leftarrow N_i \cup A$
  Return No

Let us check that the stated properties are maintained by this algorithm.

Case 1: $A \cap Y_i \neq \emptyset$. Then, by Claim 4.3, $S \setminus A \in \mathcal{I}_i$ as required. The sets Y_j and N_j are not affected, so all properties are maintained.

Case 2: $A \cap Y_i = \emptyset$ and $A \not\subseteq Y_{1-i}$. In this case, we add a to Y_i. We have $a \notin Y_{1-i}$ so the responses are No-Extensible. Furthermore, $a \notin N_i$ by property 2, and thus the responses are Consistent. $|Y_0 \cup Y_i|$ increases by at most 1 so Property 1 holds. Property 2 and the Yes-Extendibility are trivial.

Case 3: $A \cap Y_i = \emptyset$ and $A \subseteq Y_{1-i}$. In this case, we add A to N_i. It is easy to verify that Consistency, No-Extendibility, Property 1 and Property 2 are all maintained. Let us consider Yes-Extendibility. By Properties 1 and 2,

$$|N_0 \cup N_1| \leq |Y_0 \cup Y_1| \leq q.$$

So if $q < n$ then the responses are Yes-Extensible.

Since the responses are both No-Extensible and Yes-Extensible, the algorithm cannot have decided whether the two matroids have a common base. ■

The lower bound presented above is essentially tight.

Claim 4.5. *There exists a deterministic algorithm using only $n + 1$ queries that decides the matroid intersection problem for matroids in $\mathcal{M}^*$.*

Proof. For every $s \in S$, decide whether $S - s \in \mathcal{B}^*(S_1)$. In this way, the algorithm completely learns the set S_1. It must decide whether $S_0 \cap S_1 = \emptyset$. By Claim 4.3, this can be decided by testing whether $S \setminus S_1 \in \mathcal{I}(S_0)$. ■

4.1.3 A padding argument

We now build on the previous two sections and give a lower bound for matroids of any rank via a padding argument.

First we start by padding the matroids from Section 4.1.1. For any $r \geq 1$, let P be an arbitrary set such that $|P| = r - 1$ and $S \cap P = \emptyset$. Let $m = |S|$ and $n = |S \cup P| = m + r - 1$. For any $\emptyset \neq X \subseteq S$, we define the matroid $\mathrm{M}_r(X)$ as follows: it has ground set $S \cup P$ and base family $\mathcal{B}_r(X) = \{ P + x : x \in X \}$. (In matroid terminology, $\mathrm{M}_r(X)$ is obtained from $\mathrm{M}(X)$ by adding the elements in P as coloops.) This family of matroids is denoted $\mathcal{M}_r = \{ \mathrm{M}_r(X) : \emptyset \neq X \subseteq S \}$. Clearly $\mathrm{M}_r(X)$ and $\mathrm{M}_r(Y)$ have a common base if and only if $\mathrm{M}(X)$ and $\mathrm{M}(Y)$ do. Thus, the number of queries needed to solve matroid intersection for matroids in $\mathcal{M}_r$ is at least $2m - 2 = 2(n - r)$, by Theorem 4.1.

Now we consider the matroids from Section 4.1.2. Let r satisfy $0 < r < n$. Let P and S be disjoint sets with $|P| = n - r - 1$ and $|S| = r + 1$, so $|S \cup P| = n$. For any $\emptyset \neq X \subseteq S$, we define the matroid $\mathrm{M}^*_r(X)$ as follows: it has ground set $S \cup P$ and base family $\mathcal{B}^*_r(X) = \{ S - x : x \in X \}$. (In matroid terminology, the matroid $\mathrm{M}^*_r(X)$ is obtained from $\mathrm{M}^*(X)$ by adding the elements in P as loops.) This family of matroids is denoted $\mathcal{M}^*_r = \{ \mathrm{M}^*_r(X) : \emptyset \neq X \subseteq S \}$. Clearly $\mathrm{M}^*_r(X)$ and $\mathrm{M}^*_r(Y)$ have a common base if and only if $\mathrm{M}^*(X)$ and $\mathrm{M}^*(Y)$ do. Thus, the number of queries needed to solve matroid intersection for matroids in $\mathcal{M}_r$ is at least $r + 1$, by Theorem 4.4.

We summarize this discussion with the following theorem.

Theorem 4.6. *The number of independence oracle queries needed by any deterministic algorithm that solves matroid intersection for matroids with ground set size $n \geq 2$ and rank $0 < r < n$ is at least*

$$\max \{2(n - r), r + 1\}.$$

4.2 An algebraic lower bound

This section improves on Theorem 4.6 by showing an improved lower bound for matroids of rank close to $n/2$. Formally, we show the following theorem.

Theorem 4.7. *The number of independence oracle queries needed by any deterministic algorithm that solves matroid intersection for matroids with even ground set size n and*

rank $n/2+1$ *is at least* $(\log_2 3)\, n - o(n)$.

Thus by combining Theorem 4.6 and Theorem 4.7 and using padding arguments, we obtain the following result, which justifies Figure 4-1.

Corollary 4.8. *The number of independence oracle queries needed by any deterministic algorithm that solves matroid intersection is lower bounded as follows. Suppose the algorithm is given two matroids with ground set size* $n \geq 2$ *and rank* $0 < r < n$, *with* $\tilde{r} = \min\{r, n-r\}$. *Then the lower bound is*

$$\max\ \{\ 2(n-r),\ r+1,\ (\log_2 9)\tilde{r} - o(\tilde{r})\ \}\,.$$

Proof. We consider the third term. Let $\mathcal{M}$ be the family of matroids for which the lower bound of Theorem 4.7 is proven, where we choose their ground set size to be S, with $|S| = 2r-2$. Add $n-2r+2$ loops to the matroids in $\mathcal{M}$; the resulting matroids have ground set size n and rank $|S|/2+1 = r$. Then we have $\tilde{r} \geq r-2$ and, by Theorem 4.7, the lower bound on the required number of queries is

$$(\log_2 3)(2r-2) - o(r) \;=\; (\log_2 9)\tilde{r} - o(\tilde{r}).$$

If we had added $n-2r+2$ coloops instead of loops, the resulting matroids would have ground set size n and rank $\big(|S|/2+1\big) + \big(n-2r+2\big) = n-r+2$. Then we have $\tilde{r} = r-2$ and the lower bound is again

$$(\log_2 3)(2r-2) - o(r) \;=\; (\log_2 9)\tilde{r} - o(\tilde{r}).$$

This completes the proof. ■

The remainder of this section proves Theorem 4.7. A high-level overview of the proof is as follows. We describe a family of matroids that correspond to a "pointer chasing" problem. Roughly speaking, M_1 corresponds to a permutation π in the symmetric group $\mathcal{S}_n$ and M_2 corresponds to a permutation $\sigma \in \mathcal{S}_n$. Both matroids have rank $n/2+1$. The two matroids have a common base iff the cycle structure of the composition $\sigma^{-1} \circ \pi$ satisfies a certain property. The difficulty of deciding this property is analyzed using the ***communication complexity*** framework, which we introduce next. Roughly speaking, the two given matroids are anthropomorphized into two computationally unbounded players, Alice and Bob, and one analyzes

the number of bits that must be communicated between them to solve the matroid intersection problem. This yields a lower bound on the number of independence queries required by any algorithm.

A standard technique for proving lower bounds in this framework is based on the communication matrix C, which is the truth table of the function that Alice and Bob must compute. It is known that $\log_2 \operatorname{rank} C$ gives a lower bound on the number of bits which must be communicated between Alice and Bob. Since our instances are derived from the symmetric group, it is natural to use representation theory to analyze the matrix's rank. Section 4.2.5 does this by viewing the communication matrix as an operator in the group algebra. Surprisingly, we show that the matrix is diagonalizable (in Young's seminormal basis), its eigenvalues are all integers, and their precise values can be computed by considering properties of Young tableaux.

4.2.1 Communication complexity

Our lower bound uses methods from the field of communication complexity. The basics of this field are covered in the survey of Lovász [60], and further details can be found in the book of Kushilevitz and Nisan [54]. This section briefly describes the concepts that we will need.

A ***communication problem*** is specified by a function $f(X, Y)$, where X is Alice's input, Y is Bob's input, and the range is $\{0, 1\}$. A communication problem is solved by a ***communication protocol***, in which Alice and Bob send messages to each other until one of them can decide the solution $f(X, Y)$. The player who has found the solution declares that the protocol has halted, and announces the solution.

The ***deterministic communication complexity*** of f is defined to be the minimum total bit-length of the messages sent by any deterministic communication protocol for f. This quantity is denoted $D(f)$.

Nondeterminism also plays an important role in communication complexity. This model involves a third party — a ***prover*** who knows both X and Y. In a ***nondeterministic protocol*** for f, the prover produces a single certificate Z which is delivered to both Alice and Bob. (Z is a function of both X and Y). Alice and Bob cannot communicate, other than receiving Z from the prover. If $f(X, Y) = 1$, then the certificate must suffice to convince Alice and Bob of this fact (Alice sees only X and Z, Bob sees only Y and Z). Otherwise, if $f(X, Y) = 0$, no certificate should be able to fool both Alice and Bob. The ***nondeterministic communication***

complexity is defined to be the minimum length of the certificate (in bits) in any nondeterministic protocol. We denote this quantity by $N^1(f)$.

A ***co-nondeterministic protocol*** is defined analogously, reversing the roles of 1 and 0. The ***co-nondeterministic complexity*** is also defined analogously, and is denoted by $N^0(f)$.

Fact 4.9. *$N^0(f) \leq D(f)$ and $N^1(f) \leq D(f)$.*

Proof. See [54, §2.1]. Consider any deterministic communication protocol for f. Since the prover has both Alice's and Bob's inputs, it can produce a certificate containing the sequence of messages that would have been exchanged by this protocol on the given inputs. Alice and Bob can therefore use this certificate to simulate execution of the protocol, without exchanging any messages. Therefore this certificate acts both as a nondeterministic and co-nondeterministic proof. ■

Fact 4.10. *For any communication problem f, we have $D(f) = O(N^0(f) \cdot N^1(f))$.*

Proof. See [54, p20] or [60, p244]. ■

For any communication problem f, the ***communication matrix*** is a matrix $C(f)$, or simply C, whose entries are in $\{0, 1\}$, whose rows are indexed by Alice's inputs X and whose columns are indexed by Bob's inputs Y. The entries of C are $C(f)_{X,Y} = f(X, Y)$. There is a connection between algebraic properties of the matrix $C(f)$ and the communication complexity of f, as shown in the following lemma.

Fact 4.11 (Mehlhorn and Schmidt [65]). *Over any field (including the complex numbers), we have $D(f) \geq \log_2 \operatorname{rank} C(f)$.*

Proof. See [54, p13]. ■

4.2.2 Communication complexity of matroid intersection

Let us now consider the matroid intersection problem in the communication complexity framework.

Definition 4.12. *The communication problem* MATINT:

- *Alice's Input: A matroid* $\mathbf{M}_1 = (S, \mathcal{I}_1)$.
- *Bob's Input: A matroid* $\mathbf{M}_2 = (S, \mathcal{I}_2)$.
- *Output: If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *have a common base then* $\textsc{MatInt}(\mathbf{M}_1, \mathbf{M}_2) = 1$. *Otherwise, it is* 0.

Fact 4.13. $D(\textsc{MatInt})$ *gives a lower bound on the number of oracle queries made by any deterministic matroid intersection algorithm.*

Proof. See [54, Lemma 9.2]. The proof is a simulation argument: any deterministic matroid intersection algorithm which uses q independence oracle queries can be transformed into a deterministic communication protocol for MATINT that uses q bits of communication. Both Alice and Bob can independently simulate the given algorithm, and they only need to communicate whenever an oracle query is made, so the number of bits of communication is exactly q. ■

The remainder of this section focuses on analyzing the communication complexities of MATINT. Some easy observations can be made using matroids of rank one, as defined in Section 4.1.1. Recall that for two matroids $\mathbf{M}(X), \mathbf{M}(Y) \in \mathcal{M}$, they have a common base iff $X \cap Y \neq \emptyset$. Thus, for the family $\mathcal{M}$, the MATINT problem is simply the complement of the well-known ***disjointness*** problem (denoted DISJ) [54]. It is known that $D(\textsc{Disj}) \geq n$ and $N^1(\textsc{Disj}) \geq n - o(n)$. Although we will not discuss randomized complexity in any detail, it is also known [81] that the randomized communication complexity of DISJ is $\Omega(n)$, and consequently the same is true of MATINT.

Thus we have shown that $D(\textsc{MatInt}) \geq n$ and $N^0(\textsc{MatInt}) \geq n - o(n)$. In Section 4.3, we will also show that $N^1(\textsc{MatInt}) = \Omega(n)$. As it turns out, these lower bounds for N^0 and N^1 are essentially tight. To show this, we will use the matroid intersection theorem (Fact 3.1).

Lemma 4.14. $N^1(\textsc{MatInt}) \leq n$ *and* $N^0(\textsc{MatInt}) \leq n + \lfloor \log n \rfloor + 1$.

Proof. To convince Alice and Bob that their two matroids have a common base, it suffices to present them with that base B. Alice and Bob independently check that B is a base for their respective matroids. The set B can be represented using n bits, hence $N^1(\textsc{MatInt}) \leq n$.

To convince Alice and Bob that their two matroids do not have a common base, we invoke the matroid intersection theorem. The prover computes a set $A \subseteq S$

which is a minimizing set in Fact 3.1. The co-nondeterministic certificate Z consists of the set A and an integer z. Alice checks that $z = r_1(A)$. Bob checks that $z + r_2(S \setminus A) < r$. If this holds then the two matroids cannot have a common base. The length of this certificate is at most $n + \lfloor \log n \rfloor + 1$. ■

Lemma 4.14 is an unfortunate obstacle in our quest to prove a super-linear lower bound on $D(\text{MATINT})$. The fact that both the nondeterministic and co-nondeterministic communication complexities are $O(n)$ makes our task more difficult, for two reasons. First, we must use techniques that can separate the deterministic complexity from the nondeterministic complexities: we need a super-linear lower bound for $D(\text{MATINT})$ which does not imply that either $N^0(\text{MATINT})$ or $N^1(\text{MATINT})$ is super-linear (since this is false!). Second, the nondeterministic and co-nondeterministic communication complexities provably constrain the quality of any lower bound on the deterministic complexity, as shown in Fact 4.10. Thus, the communication complexity technique cannot prove a super-quadratic lower bound for the matroid intersection problem; at least, not in the present formulation.

4.2.3 The IN-SAME-CYCLE problem

One interesting category of communication problems is pointer chasing problems [10, 20, 76, 77, 80]. We now show that matroid intersection leads to an interesting pointer chasing problem.

The motivating example to keep in mind is the class of ***almost 2-regular bipartite graphs***. Let G be a graph with a bipartition of the vertices into U and V. Each vertex in U (resp., in V) has degree 2, except for two distinguished vertices $u_1, u_2 \in U$ (resp., $v_1, v_2 \in V$), which have degree 1. (So $|U| = |V|$.) The connected components of G are two paths with endpoints in $\{u_1, u_2, v_1, v_2\}$, and possibly some cycles. One can argue that G has a perfect matching iff G does not contain a path from u_1 to u_2 (equiv., from v_1 to v_2). The main idea of the argument is that odd-length paths have a perfect matching whereas even-length paths do not.

Let us now reformulate this example slightly. Let $S = U \cup V$ where $|U| = |V| = N := n/2$. Let $\mathcal{P}$ be a partition of S into pairs, where each pair contains exactly one element of U and one element of V. We can write $\mathcal{P}$ as $\{ \{u_i, v_{\pi(i)}\} : i = 1, \ldots, N \}$, where $\pi : U \to V$ is a bijection. Now $\mathcal{P}$ can be used to define a matroid. Fix

arbitrarily $1 \leq k \leq N$, and let $\mathcal{B}_k^\pi$ be the family of all B such that

$$|B \cap \{u_i, v_{\pi(i)}\}| = \begin{cases} 2 & (\text{if } i = k) \\ 1 & (\text{otherwise}). \end{cases}$$

One may verify that $\mathcal{B}_k^\pi$ is the family of bases of a partition matroid, which we denote M_k^π. Let $\mathcal{M}_k$ be the set of all such matroids (keeping k fixed, and letting π vary).

Lemma 4.15. *Let $\mathrm{M}_1^\pi \in \mathcal{M}_1$ and $\mathrm{M}_2^\sigma \in \mathcal{M}_2$. Note that $\sigma^{-1} \circ \pi$ is a permutation on U. We claim that M_1^π and M_2^σ have a common base iff elements u_1 and u_2 are in the same cycle of $\sigma^{-1} \circ \pi$.*

The proof of this lemma mirrors the argument characterizing when almost 2-regular bipartite graphs have a perfect matching. A formal proof is in Section 4.5. Let us now interpret Lemma 4.15 in the communication complexity framework.

Definition 4.16. *The* IN-SAME-CYCLE, *or* ISC, *problem:*

- *Alice's input: A permutation $\pi \in \mathcal{S}_N$.*
- *Bob's input: A permutation $\sigma \in \mathcal{S}_N$.*
- *Output: If elements 1 and 2 are in the same cycle of $\sigma^{-1} \circ \pi$, then $\mathrm{ISC}(X, Y) = 1$. Otherwise it is 0.*

We will show hardness for MATINT by analyzing ISC. First, Lemma 4.15 shows that ISC reduces to MATINT. Next, we will argue that ISC is a "hard" problem. Intuitively, it seems that Alice and Bob cannot decide the ISC problem unless one of them has learned the entire cycle containing 1 and 2, which might have length $\Omega(N)$. So it is reasonable to believe that $\Omega(N \log N)$ bits of communication are required. The remainder of this section proves the following theorem.

Theorem 4.17. *Let C denote the communication matrix for* ISC. *Then* $\operatorname{rank} C$ *equals*

$$1 + \sum_{1 \leq i \leq N-1} \sum_{1 \leq j \leq \min\{i, N-i\}} \binom{N}{i,\ j,\ N-i-j}^2 \cdot \frac{j^2 (i-j+1)^2}{N(N-1)(N-i)(N-j+1)}.$$

Corollary 4.18. $D(\mathrm{ISC}) \geq (\log_2 9)N - o(N)$. *Consequently, any deterministic algo-*

rithm solving the matroid intersection problem for matroids with rank $n/2+1$ and ground set size n must use at least $(\log_2 3)n - o(n)$ queries.

Proof. Stirling's approximation shows that

$$e\left(\frac{n}{e}\right)^n < n! < en\left(\frac{n}{e}\right)^n.$$

Thus,

$$\binom{N}{N/3,\ N/3,\ N/3} = \frac{N!}{\left((N/3)!\right)^3} \geq \frac{e\,(N/e)^N}{\left(e\,(N/3)\,(N/3e)^{N/3}\right)^3} = 3^{N-o(N)}.$$

In Theorem 4.17, considering just the term $i = j = N/3$ shows that $\operatorname{rank} C \geq 9^{N-o(N)}$. Fact 4.11 therefore implies the lower bound on $D(\mathrm{ISC})$. The lower bound for matroid intersection follows since the matroids in $\mathcal{M}_k$ have rank $n/2+1 = N+1$ and ground set size n. ■

This corollary establishes Theorem 4.7.

4.2.4 Group theory

To prove Theorem 4.17, we need to introduce several notions from group theory. We recommend Artin [2], James-Kerber [48], Naĭmark [72], Sagan [83] and Stanley [87] for a more detailed exposition of this material.

Definition 4.19. *A **representation** of a group is a homomorphism $\phi : G \to GL_d$, where GL_d denotes the general linear group of non-singular $d \times d$ matrices over $\mathbb{C}$. In other words, to every element g in G, $\phi(g)$ gives a matrix M_g such that:*

- *$M_1 = I$, where 1 is the identity element of G and I is the identity matrix.*
- *$M_g \cdot M_h = M_{gh}$.*

*The **dimension of** ϕ is defined to be d.*

Definition 4.20. *Let ϕ be a representation of G into GL_d. A subspace V of $\mathbb{C}^d$ is called **invariant** under ϕ if, for all $g \in G$, $v \in V \implies \phi(g) \cdot v \in V$. If the only two subspaces invariant under ϕ are $\{0\}$ and $\mathbb{C}^d$ itself then ϕ is called **irreducible**. An irreducible representation is often abbreviated to **irrep**.*

Fact 4.21. *For any finite group G, there are only finitely many irreps that are not isomorphic under a change of basis. Moreover, the number of non-isomorphic irreps equals the number of distinct conjugacy classes of G.*

Fact 4.22 (Maschke's Theorem). *Let G be a finite group and let ϕ be a representation of G into GL_d. Then there exist a matrix B and irreps $\rho_1, \ldots, \rho_k$ of G (possibly identical) such that, for all $g \in G$,*

$$B\phi(g)B^{-1} = \begin{pmatrix} \rho_1(g) & & \\ & \ddots & \\ & & \rho_k(g) \end{pmatrix}.$$

That is, there exists an isomorphic representation $B\phi(\cdot)B^{-1}$ which is decomposed into a direct sum of irreps.

Regular representation. Let G be an arbitrary finite group. Consider the set of formal linear combinations of elements of G, i.e., the set

$$\mathbb{C}[G] := \left\{ \sum_{g \in G} \alpha_g \cdot g \,:\, \alpha_g \in \mathbb{C} \right\}.$$

Clearly $\mathbb{C}[G]$ is a vector space, and is isomorphic to $\mathbb{C}^{|G|}$. One may define a multiplication operation on $\mathbb{C}[G]$ as follows:

$$\left(\sum_{g \in G} \alpha_g \cdot g \right) \cdot \left(\sum_{g' \in G} \beta_{g'} \cdot g' \right) = \sum_{h \in G} \left(\sum_{g \cdot g' = h} \alpha_g \beta_{g'} \right) \cdot h.$$

Thus $\mathbb{C}[G]$ is an algebra, known as the ***group algebra*** of G.

Define an action of G on $\mathbb{C}[G]$ as follows. For $h \in G$, let

$$h \cdot \left(\sum_{g \in G} \alpha_g \cdot g \right) = \sum_{g \in G} \alpha_g \cdot (g \cdot h).$$

Thus, the action of h amounts to a permutation of the coordinates of $\mathbb{C}^{|G|}$, and

therefore it can be represented as a permutation matrix

$$R(h)_{g,g'} = \begin{cases} 1 & \text{if } g' \cdot h = g \\ 0 & \text{otherwise.} \end{cases}$$

The mapping from elements of G to the matrices R is a homomorphism of G into $GL_{|G|}$, and hence it is a representation of G, known as the ***regular representation.***

Fact 4.23. *Let G be a finite group. By Fact 4.22, the regular representation decomposes into a direct sum of irreps $\{\rho_1, \ldots, \rho_k\}$. Every irrep of G appears in this list, and moreover the number of occurrences in this list of an irrep ρ_i equals the dimension of ρ_i.*

Symmetric group. We now discuss the symmetric group $\mathcal{S}_N$ of all permutations on $[N]$, i.e., bijections from $[N]$ to $[N]$ under the operation of function composition.

Let $\pi \in \mathcal{S}_N$. The ***cycle type*** of π is a sequence of integers in which the number of occurrences of k equals the number of distinct cycles of length k in π. Without loss of generality, we may assume that this sequence is in non-increasing order. Thus, the cycle type of π is a partition of N, which is defined as a non-increasing sequence $\lambda = (\lambda_1, \ldots, \lambda_\ell)$ of positive integers such that $N = \sum_{i=1}^{\ell} \lambda_i$. The value ℓ is called the ***length*** of λ, and it is also denoted $\ell(\lambda)$. The notation $\lambda \vdash N$ denotes that the sequence λ is a partition of N.

Let $\mathcal{C}(\lambda) \subseteq \mathcal{S}_N$ be the set of all permutations with cycle type $\lambda \vdash N$. The set $\mathcal{C}(\lambda)$ is a conjugacy class of $\mathcal{S}_N$. Moreover, every conjugacy class of $\mathcal{S}_N$ is obtained in this way, so the number of conjugacy classes of $\mathcal{S}_N$ equals the number of partitions of N. Thus, by Fact 4.21, we see that the non-isomorphic irreps of $\mathcal{S}_N$ can be indexed by the partitions of N. Henceforth, the irreps of $\mathcal{S}_N$ will be denoted ρ_λ where $\lambda \vdash N$.

A ***Ferrers diagram*** of $\lambda \vdash N$ is a left-aligned array of boxes in the plane for which the i^{th} row contains λ_i boxes. An example is shown in Figure 4-2 (a). A ***Young tableau*** of shape λ is a bijective assignment of the integers in $[N]$ to the boxes of the Ferrers diagram for λ. An example is shown in Figure 4-2 (b). A ***standard Young tableau,*** or ***SYT,*** is one in which

- for each row, the values in the boxes increase from left to right, and
- for each column, the values in the boxes increase from top to bottom.

An example is shown in Figure 4-2 (c).

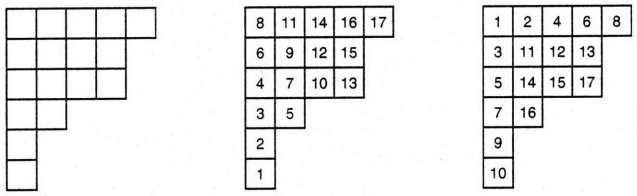


Figure 4-2: (a) The Ferrers diagram for the partition $(5, 4, 4, 2, 1, 1) \vdash 17$. (b) A Young tableau. (c) A standard Young tableau.

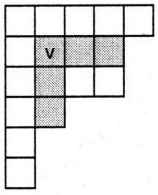


Figure 4-3: A box v and its hook h_v.

Let $\lambda \vdash N$. Let v be a box in the Ferrers diagram of λ. The ***hook*** of box v, denoted h_v, is the set of boxes in the same row as v but to its right or in the same column as v but beneath it (including v itself). This is illustrated in Figure 4-3.

Fact 4.24 (Hook Length Formula). *The number of SYT of shape λ is denoted f_λ, and has value*

$$f_\lambda := \frac{N!}{\prod_v |h_v|},$$

where the product is over all boxes in the Ferrers diagram for λ.

Fact 4.25. *The dimension of irrep ρ_λ equals f_λ, the number of SYT of shape λ. Thus Fact 4.24 provides a formula for the dimension of ρ_λ.*

There exist several canonical ways of defining the irrep associated to partition λ, since a change of basis produces an isomorphic representation. In this thesis, we will fix ***Young's seminormal basis*** [48] as the specific basis in which each irrep is presented. The formal definition of this basis is not crucial for us, but we will need some of its properties.

First, we introduce some notation. Let Y_λ denote the irrep corresponding to

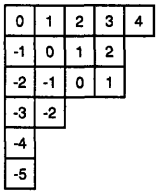


Figure 4-4: The "content" of all boxes in this Ferrers diagram.

partition λ in Young's seminormal basis. For any $\pi \in \mathcal{S}_N$, the notation $Y_\lambda(\pi)$ denotes the matrix associated with π by this irrep. For any set $S \subseteq \mathcal{S}_N$, let $Y_\lambda(S) = \sum_{\pi \in S} Y_\lambda(\pi)$.

For $1 \leq j \leq N$, the j^{th} ***Jucys-Murphy element*** is the member of the group algebra defined by $J_j = \sum_{1 \leq i < j} (i, j)$. (Here, (i, j) denotes a transposition in $\mathcal{S}_N$.) For convenience, we may also view J_j as a subset of $\mathcal{S}_N$, namely the set of $j - 1$ transpositions which appear with non-zero coefficient in J_j.

For a Ferrers diagram of shape λ, the ***content*** of the box (a, b) (i.e., the box in row a and column b) is the integer $b - a$. This is illustrated in Figure 4-4; note that the content values are constant on each negative-sloping diagonal. For any standard Young tableau t and $1 \leq j \leq N$, define $\text{cont}(t, j)$ to be the content of the box occupied by element j in tableau t.

Fact 4.26. *$Y_\lambda(J_j)$ is a diagonal matrix and the diagonal entries are $Y_\lambda(J_j)_{t,t} = \text{cont}(t, j)$, where t is a tableau of shape λ.*

A proof of this fact can be found in the book of James and Kerber [48].

4.2.5 Analysis of IN-SAME-CYCLE

In this section, we compute the rank of the communication matrix for the ISC problem, and thereby prove Theorem 4.17. Surprisingly, we will show that this matrix is diagonalizable, and that the values of those diagonal entries (i.e., the spectrum) are integers that can be precisely computed.

Overview of Proof. Our argument proceeds as follows.

- **Step 1.** The matrix C can be written as a sum of matrices in the regular

representation.

- **Step 2.** There exists a change-of-basis matrix which block-diagonalizes the matrices of the regular representation (i.e., decomposes them into irreps). Thus C can also be block-diagonalized.
- **Step 3.** The blocks of C can be expressed as a polynomial in the matrices corresponding to the Jucys-Murphy elements. Thus each block is actually a diagonal matrix (if the change-of-basis matrix is chosen properly).
- **Step 4.** The diagonal entries of each block (i.e., eigenvalues of C) are given by a polynomial in the content values, so they can be explicitly computed. The rank of C is simply the number of non-zero eigenvalues, so a closed form expression for the rank can be given.

Step 1. Let $\pi \in \mathcal{S}_N$ be the permutation corresponding to Alice's input and let $\sigma \in \mathcal{S}_N$ correspond to Bob's input. Define $\mathcal{K}_N$, or simply $\mathcal{K}$, to be

$$\mathcal{K}_N = \{ \tau \in \mathcal{S}_N : 1 \text{ and } 2 \text{ are in the same cycle of } \tau \}.$$

Note that $\mathcal{K}$ is closed under taking inverses: $\pi \in \mathcal{K} \implies \pi^{-1} \in \mathcal{K}$. Recall the definition of the communication matrix C:

$$C_{\pi,\sigma} = \begin{cases} 1 & \text{if } \sigma^{-1} \circ \pi \in \mathcal{K}, \\ 0 & \text{otherwise.} \end{cases}$$

This leads to the following easy lemma.

Lemma 4.27. *$C = \sum_{\tau \in \mathcal{K}} R(\tau)$, where $R(\tau)$ denotes a matrix of the regular representation.*

Proof. Let $\rho = \sigma^{-1} \circ \pi$, implying that $\pi = \sigma \circ \rho$. Clearly ρ is the unique permutation with this property. Thus $R(\tau)_{\pi,\sigma} = 1$ iff $\tau = \rho$. Thus, the entry in row π and column σ of $\sum_{\tau \in \mathcal{K}} R(\tau)$ is 1 if $\rho \in \mathcal{K}$ and 0 otherwise. This matches the definition of C. ■

Step 2. Now let B be the change-of-basis matrix which decomposes the regular representation into irreps, as mentioned in Fact 4.22. We will analyze the rank of

C by considering the contribution from each irrep. We have

$$\operatorname{rank} C \;=\; \operatorname{rank} B\,C\,B^{-1} \;=\; \operatorname{rank}\Big(\sum_{\tau\in\mathcal{K}} B\,R(\tau)\,B^{-1}\Big) \;=\; \sum_{\lambda\vdash N} f_\lambda \cdot \operatorname{rank} Y_\lambda(\mathcal{K}), \quad (4.1)$$

where the second equality follows from Lemma 4.27. To see the third equality, recall that each $B\,R(\tau)\,B^{-1}$ is decomposed into blocks of the form $Y_\lambda(\tau)$ (see Fact 4.23), so each block of BCB^{-1} is of the form $Y_\lambda(\mathcal{K})$. Furthermore, each irrep λ appears f_λ times (see Fact 4.25.)

Step 3. The following lemma gives the reason that the communication matrix for ISC can be analyzed so precisely. It gives a direct connection between the ISC problem and the Jucys-Murphy elements.

Lemma 4.28. $\sum_{\pi\in\mathcal{K}} \pi \;=\; J_2 \cdot \prod_{j=3}^{N}(1+J_j)$, *where* 1 *denotes the identity permutation.*

Proof. The proof is by induction on N, the trivial case being $N = 2$. So let $N > 2$. For any $\pi \in \mathcal{K}_{N-1}$ and transposition (i, N), we have $\pi \circ (i, N) \in \mathcal{K}_N$. Conversely, for any $\pi \in \mathcal{K}_N$, there is a unique way to obtain π as a product of $\pi' \in \mathcal{K}_{N-1}$ and a transposition (i, N), by taking $i = \pi^{-1}(N)$ and $\pi' = \pi \circ (i, N)$ (restricted to S_{N-1}). ■

Here is a simple, but interesting, corollary of this lemma.

Corollary 4.29. $|\mathcal{K}_N| = |S_N|/2$. *In other words, for any* π,

$$\Pr_\sigma\left[\,\mathrm{ISC}(\pi,\sigma) = 1\,\right] \;=\; 1/2,$$

where σ *is chosen uniformly from* S_N.

Proof. Viewing the Jucys-Murphy elements as sets, we have $|J_i| = i - 1$. Since the permutations arising in the product $J_2 \cdot \prod_{j=3}^{N}(1 + J_j)$ are distinct, we have $|\mathcal{K}_N| = 1 \cdot \prod_{j=3}^{N} j = N!/2$. ■

Lemma 4.28 shows that the sum $\sum_{\pi\in\mathcal{K}} \pi$ can be expressed as a polynomial in the Jucys-Murphy elements. In other words, for every $\lambda \vdash N$, the matrix $Y_\lambda(\mathcal{K})$ can be expressed as a polynomial in the matrices $\{\, Y_\lambda(J_j) \;:\; 2 \le j \le N \,\}$. It follows directly from Fact 4.26 that $Y_\lambda(\mathcal{K})$ is diagonal. Furthermore, we can determine the diagonal entries explicitly. For every SYT t of shape λ, the corresponding diagonal

entry of $Y_\lambda(\mathcal{K})$ satisfies the expression

$$Y_\lambda(\mathcal{K})_{t,t} \;=\; Y_\lambda(J_2)_{t,t} \cdot \prod_{j=3}^{N} \big(1 + Y_\lambda(J_j)_{t,t}\big). \tag{4.2}$$

As mentioned above, the blocks of $B\,C\,B^{-1}$ are all of the form $Y_\lambda(\mathcal{K})$. Thus $B\,C\,B^{-1}$ is actually diagonal, and Eq. (4.2) completely determines the spectrum of C, since the values $Y_\lambda(J_j)_{t,t}$ are known (see Fact 4.26).

Step 4. In the remainder of this section, we will analyze Eq. (4.2) more closely. Our main goal is to determine when its value is non-zero. This holds whenever $Y_\lambda(J_2)_{t,t} \neq 0$ and $Y_\lambda(J_j)_{t,t} \neq -1$ for all $j \geq 3$. By Fact 4.26, $Y_\lambda(J_2)_{t,t} = 0$ only when 2 lies on the main diagonal of t, which is impossible in any SYT. Similarly, $Y_\lambda(J_j)_{t,t} = -1$ only when j lies on the first subdiagonal. So we have the following fact, which is crucial to the analysis. For an SYT t,

$$Y_\lambda(\mathcal{K})_{t,t} \neq 0 \iff \text{in tableau } t\text{, all values } j \geq 3 \text{ avoid the first subdiagonal.} \tag{4.3}$$

Let us now consider three cases.

Case 1: $\lambda_3 > 1$. Fix an arbitrary SYT t of shape λ. The box in position $(3,2)$ (row 3, column 2) of t contains some value $j \geq 6$. Since this box is on the first subdiagonal, Eq. (4.3) shows that $Y_\lambda(\mathcal{K})_{t,t} = 0$.

Case 2: $\lambda_2 = 0$, i.e., $\lambda = (N)$. There is a unique SYT of shape λ, in which every box $(1,j)$ contains j. Thus $Y_\lambda(J_j) = j - 1$ for all j, so Eq. (4.2) shows that the unique entry of $Y_\lambda(\mathcal{K})$ has value $N!/2$.

Case 3: $\lambda_2 \geq 1$ and $\lambda_3 \leq 1$. In the Ferrers diagram of shape λ, only the box $(2,1)$ is on the first subdiagonal. Consider now an SYT t of shape λ. If the box $(2,1)$ contains $j \geq 3$ then $Y_\lambda(\mathcal{K})_{t,t} = 0$ by Eq. (4.3).

On the other hand, if the box $(2,1)$ contains the value 2 then all values $j \geq 3$ avoid the first subdiagonal, implying that $Y_\lambda(\mathcal{K})_{t,t} \neq 0$. In fact, the precise value of $Y_\lambda(\mathcal{K})_{t,t}$ can be determined. Since the value 2 is in box $(2,1)$ we have $Y_\lambda(J_2)_{t,t} = -1$. The multiset $\{\, Y_\lambda(J_j)_{t,t} : j \geq 3 \,\}$ is simply the multiset of all content values in boxes excluding $(1,1)$ and $(2,1)$. Let B denote this set of

$N-2$ boxes. Then

$$\begin{aligned} Y_\lambda(\mathcal{K})_{t,t} &= Y_\lambda(J_2)_{t,t} \cdot \prod_{j=3}^{N} \left(1 + Y_\lambda(J_j)_{t,t}\right) \\ &= -\prod_{(a,b)\in B} (1+b-a) \\ &= \lambda_1! \cdot (\lambda_2 - 1)! \cdot (N - \lambda_1 - \lambda_2)! \cdot (-1)^{N-\lambda_1-\lambda_2+1} \end{aligned}$$

We have now computed the entire spectrum of C. The remaining task is to compute the rank (i.e., count the number of non-zero eigenvalues). As argued above, any shape λ with $\lambda_3 > 1$ contributes zero to the rank, and the shape $\lambda = (N)$ contributes exactly 1. It remains to consider shapes with $\lambda_2 \geq 1$ and $\lambda_3 \leq 1$. As argued above, the number of non-zero diagonal entries in $Y_\lambda(\mathcal{K})$ equals the number of SYT in which box $(2,1)$ contains the value 2; let us denote this quantity by g_λ. Furthermore, there are precisely f_λ copies of the block corresponding to shape λ (by Fact 4.25). Thus,

$$\operatorname{rank} C = 1 + \sum_{\substack{\lambda \text{ s.t.} \\ \lambda_2 \geq 1 \text{ and } \lambda_3 \leq 1}} f_\lambda \cdot g_\lambda. \tag{4.4}$$

The value of this expression is obtained by the following lemma.

Lemma 4.30. *Let $\lambda \vdash N$ satisfy $\lambda_2 \geq 1$ and $\lambda_3 \leq 1$. Then*

$$\begin{aligned} f_\lambda &= \binom{N}{\lambda_1,\, \lambda_2,\, N-\lambda_1-\lambda_2} \cdot \frac{\lambda_2\,(\lambda_1 - \lambda_2 + 1)}{(N-\lambda_1)\,(N-\lambda_2+1)}. \\ g_\lambda &= \binom{N}{\lambda_1,\, \lambda_2,\, N-\lambda_1-\lambda_2} \cdot \frac{\lambda_2\,(\lambda_1 - \lambda_2 + 1)}{N(N-1)}. \end{aligned}$$

Substituting into Eq. (4.4) yields

$$1 + \sum_{1 \leq \lambda_1 \leq N-1} \; \sum_{1 \leq \lambda_2 \leq \min\{\lambda_1, N-\lambda_1\}} \binom{N}{\lambda_1,\, \lambda_2,\, N-\lambda_1-\lambda_2}^2 \cdot \frac{\lambda_2^2\,(\lambda_1 - \lambda_2 + 1)^2}{N(N-1)(N-\lambda_1)(N-\lambda_2+1)}.$$

This concludes the proof of Theorem 4.17.

4.3 Paving matroids

In this section, we introduce "one-alternation" matroid intersection algorithms, which first M_1, then query M_2, but do not again query M_1. We show that any such algorithm requires $2^{n-o(n)}$ queries to solve matroid intersection. This implies another linear lower bound for ordinary matroid intersection algorithms.

Our arguments are based on the use of paving matroids, which we now introduce. To do so, we first describe another operation on matroids which we call ***puncturing***, although this is probably not standard terminology.

Lemma 4.31. *Let $M = (S, \mathcal{B})$ be a matroid. Let $B \in \mathcal{B}$ be a base such that, for all $A \subseteq S$ with $|A| = |B|$ and $|A \oplus B| = 2$, we have $A \in \mathcal{B}$. Then $(S, \mathcal{B} - B)$ is also a matroid.*

Proof. Let r be the rank function of M. Define the function $\tilde{r} : S \to \mathbb{N}$ as follows.

$$\tilde{r}(A) = \begin{cases} r(A) - 1 & \text{if } A = B \\ r(A) & \text{otherwise} \end{cases}$$

We now claim that $\tilde{r}$ is the rank function of the matroid $(S, \mathcal{B} - B)$. To show this, it suffices to show that it is submodular, i.e., satisfies

$$\tilde{r}(A) + \tilde{r}(B) \geq \tilde{r}(A \cup B) + \tilde{r}(A \cap B) \qquad \forall\, A, B \subseteq S.$$

It is known [95] [86, Theorem 44.1] that this is equivalent to

$$\tilde{r}(A + a) + \tilde{r}(A + b) \geq \tilde{r}(A \cup \{a, b\}) + \tilde{r}(A) \qquad \forall A \subseteq S \text{ and } \forall a, b \in S \setminus A.$$

Since $\tilde{r}$ differs from r only in that $\tilde{r}(B) = r(B) - 1$, it suffices to verify whether

$$r(B) + r(B - j + i) \stackrel{?}{\geq} r(B + i) + r(B - j).$$

We have

$$\tilde{r}(B) + \tilde{r}(B - j + i) = (|B| - 1) + |B|,$$

by definition of $\tilde{r}$ and since $|B \oplus (B - j + i)| = 2$. Also,

$$\tilde{r}(B + i) + \tilde{r}(B - j) = |B| + (|B| - 1)$$

since B is a base. Thus the desired inequality is satisfied (with equality). ■

Now let S be a ground set of cardinality n, where n is even. Let $\mathbf{M} = (S, \mathcal{B})$ be the uniform matroid of rank $n/2$. Let $\mathcal{C}^* \subseteq 2^S$ be a code of minimum distance 4 for which all codewords have weight $n/2$. That is, $\mathcal{C}^* \subset \mathcal{B}$, and for all $A, B \in \mathcal{C}$ we have $|A \oplus B| \geq 4$. A greedy code construction shows that we may take

$$|\mathcal{C}^*| \geq \binom{n}{n/2}/n^2 = 2^{n-o(n)}.$$

For any subcode $\mathcal{C} \subseteq \mathcal{C}^*$, we obtain a new matroid by puncturing $\mathbf{M}$ at every set $C \in \mathcal{C}$. Formally, we define $\mathbf{P}_\mathcal{C} = (S, \mathcal{B} \setminus \mathcal{C})$. Lemma 4.31 shows that $\mathbf{P}_\mathcal{C}$ is indeed a matroid. Such matroids are known as ***paving matroids*** [53] [94, §16.6].

Suppose that Alice is given a matroid $\mathbf{P}_\mathcal{C}$ where $\mathcal{C} \subseteq \mathcal{C}^*$ and Bob is given a matroid $\mathbf{M}_B = (S, \{B\})$ where $B \in \mathcal{C}^*$. It is clear that $\mathbf{P}_\mathcal{C}$ and $\mathbf{M}_B$ have a common base iff $B \notin \mathcal{C}$. This shows a connection to the INDEX problem in communication complexity, in which Alice is given a vector $x \in \{0,1\}^m$ and Bob is given an index $i \in [m]$. Their task is to compute the value x_i. The INDEX problem reduces to matroid intersection in the following way. First, we identify $\mathcal{C}^*$ with $[m]$. Alice, given x, constructs the corresponding subset $\mathcal{C} \subseteq \mathcal{C}^*$, and the matroid $\mathbf{P}_\mathcal{C}$. Bob, given i, Bob constructs the corresponding set $B \in \mathcal{C}^*$ and $\mathbf{M}_B$. We have $x_i = 1$ precisely when $\mathbf{P}_\mathcal{C}$ and $\mathbf{M}_B$ have a common base.

This reduction implies a few results. The first result relates to ***one-round communication protocols***, in which Alice can send messages to Bob, but Bob cannot reply to Alice. These protocols correspond to "one-alternation algorithms" for matroid intersection: algorithms which first make some number of queries to $\mathbf{M}_1$, then make some number of queries to $\mathbf{M}_2$, then halt without querying $\mathbf{M}_1$ again.

Lemma 4.32. *Any (randomized or deterministic) one-alternation matroid intersection algorithm must perform $2^{n-o(n)}$ queries to $\mathbf{M}_1$.*

Proof. It is known [54] that any randomized or deterministic one-round protocol for the INDEX problem must use $\Theta(m)$ bits of communication. It follows that the communication complexity of any one-round protocol for MATINT is $\Theta(|\mathcal{C}^*|) = 2^{n-o(n)}$. The desired result then follows by a simulation argument similar to the one in Fact 4.13. ■

Lemma 4.32 yields yet another linear lower bound on the number of queries

needed by any matroid intersection algorithm, even randomized algorithms. This result is a consequence of the following fact.

Fact 4.33. *The deterministic (multi-round) communication complexity of any function f is at least the logarithm (base 2) of the deterministic one-round communication complexity. This also holds for randomized protocols.*

Proof. See Kushilevitz and Nisan [54, p49]. ■

Finally, it holds that $N^0(\text{INDEX}) \geq \log m$ and $N^1(\text{INDEX}) \geq \log m$. (This follows via a trivial reduction from the EQ and NEQ functions on $\log m$ bits; these functions are defined and analyzed in Kushilevitz and Nisan [54].) Our reduction therefore shows that $N^0(\text{MATINT}) \geq n - o(n)$ and $N^1(\text{MATINT}) \geq n - o(n)$.

4.4 Discussion

Queries vs Communication. Can one prove better lower bounds by directly analyzing query complexity rather than resorting to communication complexity? It is conceivable that matroid intersection requires $\Omega(nr^{1.5})$ queries but $D(\text{MATINT}) = O(n)$. In Section 4.1, we presented an analysis of the query complexity for very elementary matroids. Extending such an analysis to general matroids seems quite difficult as the independence oracle queries are very powerful compared to queries that have been successfully analyzed in other work, e.g., Rivest and Vuillemin [82].

IN-SAME-CYCLE. Section 4.2 analyzed the ISC problem, using a rank argument to lower bound $D(\text{ISC})$. We conjecture that the rank lower bound is weak for this problem, and that actually $D(\text{ISC}) = \omega(n)$ holds. This seems difficult to prove, due to the paucity of techniques for proving gaps between the deterministic and non-deterministic complexities.

We were able to show an $\Omega(n \log n)$ lower bound on the communication complexity of (randomized or deterministic) *one-round* communication protocols for this problem. We have also shown that $N^0(\text{ISC}) = \Omega(n)$ and $N^1(\text{ISC}) = \Omega(n)$.

The definition of ISC involved a partition $\mathcal{P}$ of a ground set $S = U \cup V$ into pairs such that each pair has exactly one element of U and one of V. This "bipartite restriction" of $\mathcal{P}$ allows us to draw a connection to permutations, and consequently to the representation theory of S_n. However, from a matroid perspective,

the assumption is unnecessary. We could have defined the ISC problem simply to involve a partition $\mathcal{P}$ of the ground set S into pairs, without respecting any bipartition. This definition does not result in a connection to $\mathcal{S}_n$, but rather to the ***Brauer algebra*** [3, 79], whose representation theory is also well-studied. However, we have shown that the rank of the resulting communication matrix is only $2^{O(n)}$.

Are there other families of matroids for which matroid intersection reduces to a permutation problem that can be analyzed by similar techniques? Could this lead to stronger lower bounds? We were unable to find other interesting families of matroids which give a clean connection to Jucys-Murphy elements, as in Lemma 4.28. However, we did find a different approach to analyzing the ISC problem, using ***characters*** rather than directly computing the spectrum. We precisely computed the number of non-zero characters using tools from the theory of symmetric functions [87]. It is possible that this approach may be less brittle than the approach using Jucys-Murphy elements, and might allow a broader class of problems to be analyzed.

Raz and Spieker. Our arguments in Section 4.2 are inspired by the work of Raz and Spieker [80], who used representation theory to analyze a similar pointer chasing problem. Define $\mathcal{L}$ to be the set of all permutations in $\mathcal{S}_N$ whose cycle structure consists of a single cycle of length N. Raz and Spieker analyze the communication complexity of deciding whether $\sigma^{-1} \circ \pi \in \mathcal{L}$, where Alice has $\pi \in \mathcal{S}_N$ and Bob has $\sigma \in \mathcal{S}_N$. Their analysis is somewhat easier than ours because $\mathcal{L}$ is a conjugacy class of $\mathcal{S}_N$ and the communication matrix is in the center of the commutant algebra of $\mathcal{S}_N$. An immediate consequence is that the communication matrix is diagonalizable.

Interestingly, their result can easily be recovered using our framework of Jucys-Murphy elements. We observe that an analog of Lemma 4.28 holds for their problem, namely $\prod_{j=2}^{N} J_j = \sum_{\pi \in \mathcal{L}} \pi$. Thus, for any $\lambda \vdash N$,

$$Y_\lambda(\mathcal{L})_{t,t} = \prod_{j=2}^{N} Y_\lambda(J_j)_{t,t}, \quad \text{for all SYT } t \text{ of shape } \lambda.$$

Thus $Y_\lambda(\mathcal{L})_{t,t} \neq 0$ iff in tableau t, every value $j \geq 2$ avoids the main diagonal. This clearly holds iff $\lambda_2 \leq 1$. Furthermore, the precise value of $Y_\lambda(\mathcal{L})_{t,t}$ can be

determined using content values, as we have done in Section 4.2.5.

We remark that the work of Raz and Spieker has different motivations than our work. They compute the rank of the communication matrix in order to show that the rank lower bound can be much smaller than the non-deterministic complexity (by a $\log \log$ factor). In our case, the non-deterministic complexities are both known to be $n + o(n)$, but we show that the rank lower bound is strictly larger than the non-deterministic complexities.

Stronger communication lower bounds. How might one prove stronger communication lower bounds for MATINT? Can one prove an $\omega(n \log n)$ lower bound? This is certainly not possible for partition matroids, laminar matroids, or graphic matroids. One can show that the number of such matroids is $2^{O(n \log n)}$, and consequently MATINT can be decided using $O(n \log n)$ bits of communication. (Alice just needs to describe her matroid to Bob.) The number of paving matroids is enormous — $2^{2^{O(n)}}$ — but it is not clear how they can be used to obtain a strong lower bound because any two paving matroids always have a common base, so the decision problem is trivial. Another important family of matroids is the ***binary matroids***, which are linear matroids representable over $\mathbb{F}_2$. The number of such matroids is $2^{O(n^2)}$, as can be seen by considering matrices of the form $\begin{pmatrix} I & M \end{pmatrix}$, where I is the identity matrix and M is arbitrary. (See Wild [96] for a more precise result.) The number of transversal matroids is also $2^{O(n^2)}$, as was shown by Brylawski [7], by a similar argument. It is possible that one could show that $D(\text{MATINT}) = \Omega(n^2)$ by using either binary or transversal matroids.

Bounded-Alternation Matroid Intersection Algorithms. In Section 4.3, we defined the notion of one-alternation algorithms for matroid intersection, and proved that such algorithms must perform $2^{n-o(n)}$ queries. The definition generalizes in the natural way to algorithms with only k alternations. Can one prove a query lower bound for k-alternation matroid intersection algorithms? Is it true that $2^{\Omega(n)}$ queries are required for any constant k?

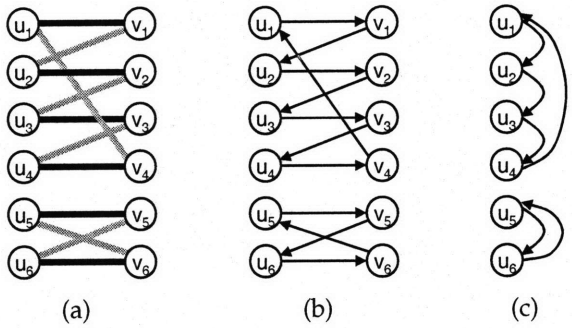


Figure 4-5: (a) The bipartite graph G. Black edges represent the parts of π and grey edges represent the parts of σ. (b) The black edges are oriented from U to V and the grey edges are oriented from V to U. Thus, we may think of the edges from V to U as representing σ^{-1}. (c) By contracting each edge $(u_i, \pi(u_i))$, we obtain a permutation on U, namely $\sigma^{-1} \circ \pi$. There is a bijection between cycles in G and cycles of $\sigma^{-1} \circ \pi$.

4.5 Proofs

Proof of Lemma 4.15

Recall that π and σ can be viewed as partitions of S for which each part has size 2. We construct a bipartite (multi-)graph G with left-vertices U, right-vertices V, and edges given by the parts of π and σ. Each vertex of G has degree 2, and therefore each connected component of G is a cycle. Let $C_1, \ldots, C_\ell$ denote the edge sets of these ℓ cycles, and assume that cycle C_1 contains vertex u_1. If F is a set of edges, let $U(F)$ and $V(F)$ respectively denote the U-vertices and V-vertices covered by F. An edge which corresponds to a part of π is called a π-edge; otherwise, it is called a σ-edge. An illustration is given in Figure 4-5.

Consider traversing the cycles of G as follows. Begin with a vertex u_a. Follow the π-edge incident with u_a, arriving at a vertex v_a. Next, follow the σ-edge incident with v_a, arriving at a vertex u_b. Repeat this process until returning to u_a. The U-vertices traversed in this process form a cycle of $\sigma^{-1} \circ \pi$. Furthermore, this process gives a bijection between cycles of G and cycles of $\sigma^{-1} \circ \pi$, and the ordering of vertices within each cycle is preserved by this bijection.

$\Longleftarrow$: For consistency with our example, let us consider u_3 instead of u_2. So suppose that u_1 and u_3 are in the same cycle of $\sigma^{-1} \circ \pi$. We will show that M_1^π and

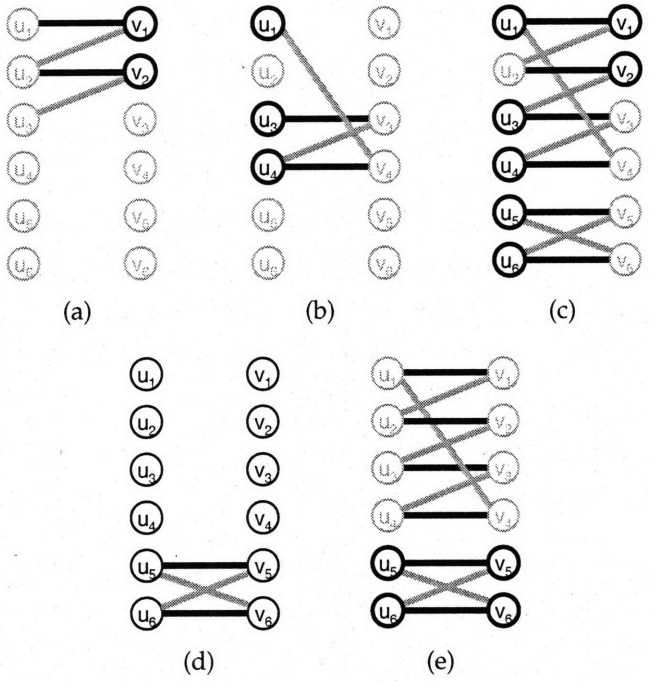


Figure 4-6: (a) The path P. The vertices $V(P)$ are shown in bold. (b) The path P'. The vertices $U(P')$ are shown in bold. (c) The vertices B are shown in bold. (d) The cycle C. (e) The set of vertices A is shown in bold and $\overline{A}$ is shown in grey.

M_3^σ have a common base. By our earlier remarks, u_1 and u_3 are also in the same cycle of G, namely C_1. This cycle can be decomposed into two edge-disjoint paths both of which have endpoints u_1 and u_3. Let P be the path which contains the π-edge incident with u_1, and let P' be the other path. Note that P must also contain the σ-edge incident with u_3. Define

$$B = V(P) \cup U(P') \cup \bigcup_{i=2}^{\ell} U(C_i).$$

Figure 4-6 (a)-(c) give an illustration.

Consider an arbitrary part of π, say $\{u_i, v_{\pi(i)}\}$. The following cases are easy to verify.

Case 1: $i = 1$. Then u_1 and $v_{\pi(1)}$ are both in B.

Case 2: $i > 1$ and $\{u_i, v_{\pi(i)}\}$ is on path P. Then $u_i \notin B$ and $v_{\pi(i)} \in B$.

Case 3: $i > 1$ and $\{u_i, v_{\pi(i)}\}$ is not on path P. Then $u_i \in B$ and $v_{\pi(i)} \notin B$.
It follows that B is a base for M_1^π. A symmetric argument shows that B is also a base for M_3^σ.

$\Longrightarrow$: For consistency with our example, let us consider u_5 instead of u_2. Suppose that u_1 and u_5 are not in the same cycle of $\sigma^{-1} \circ \pi$. We will show that M_1^π and M_3^σ do not have a common base. Let C be the cycle of G containing u_5. Let A be the set of all vertices covered by C. Figure 4-6 (d)-(e) give an illustration. Then we claim that $\mathrm{rank}_{\mathrm{M}_1^\pi}(A) = |A|/2$. This follows because:

- $|A \cap \{u_i, v_{\pi(i)}\}|$ is either 0 or 2.
- If $|A \cap \{u_i, v_{\pi(i)}\}| = 0$ then these vertices obviously contribute 0 to $\mathrm{rank}_{\mathrm{M}_1^\pi}(A)$.
- If $|A \cap \{u_i, v_{\pi(i)}\}| = 2$ then these vertices contribute only 1 to $\mathrm{rank}_{\mathrm{M}_1^\pi}(A)$ by definition of M_1^π. (Note that A does not intersect $\{u_1, v_{\pi(1)}\}$.)

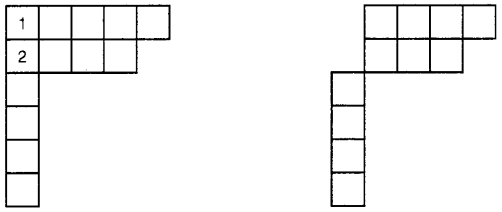
Similarly, $\mathrm{rank}_{\mathrm{M}_5^\sigma}(\overline{A}) = |\overline{A}|/2$. Thus $\mathrm{rank}_{\mathrm{M}_1^\pi}(A) + \mathrm{rank}_{\mathrm{M}_5^\sigma}(\overline{A}) = n$, although both M_1^π and M_5^σ have rank $n+1$. The weak direction of the matroid intersection theorem (Fact 3.1) therefore implies that M_1^π and M_5^σ cannot have a common base.

Proof of Lemma 4.30

Let $\lambda \vdash n$ and $\mu \vdash n$ be such that $\mu_i \leq \lambda_i$ for all i. Consider the set of boxes that are contained in the Ferrers diagram of λ but not of μ. This set is called a ***skew shape***, and is denoted $\lambda \setminus \mu$. The definition of a standard Young tableau generalizes to skew shapes in the obvious way.

We seek to understand g_λ, the number of SYT of shape λ in which the value 2 is in box $(2,1)$. (Note that the box $(1,1)$ contains the value 1 in any SYT.) Equivalently, g_λ equals the number of SYT of skew shape $\lambda \setminus \mu$ where μ is the partition $(1,1)$. This is illustrated in Figure 4-7.

The SYT of shape $\lambda \setminus \mu$ are easily enumerated. First, one chooses the elements from $\{3, \ldots, n\}$ which will occupy the first two rows. (The remaining elements will occupy the vertical bar, i.e., the rows other than the first two.) There are $\binom{n-2}{\lambda_1+\lambda_2-2}$ ways to choose these elements. If the final arrangement is to be an SYT, then there is a single way to arrange the remaining elements in the vertical bar, i.e., increasing downwards. It remains to enumerate the number of SYT on the first two rows. It



(a) Shape λ (b) Shape $\lambda \setminus \mu$

Figure 4-7: The SYT of shape λ in which box $(2,1)$ contains element 2 correspond to SYT of shape $\lambda \setminus \mu$.

follows from the Hook Length Formula (Fact 4.24) that the number of SYT of shape (a,b) is $\binom{a+b}{a} - \binom{a+b}{a+1}$.

Thus a simple manipulation shows that

$$\begin{aligned} g_\lambda &= \binom{n-2}{\lambda_1+\lambda_2-2} \cdot \left(\binom{\lambda_1+\lambda_2-2}{\lambda_1-1} - \binom{\lambda_1+\lambda_2-2}{\lambda_1} \right) \\ &= \binom{n}{\lambda_1,\, \lambda_2,\, n-\lambda_1-\lambda_2} \cdot \frac{\lambda_2\,(\lambda_1-\lambda_2+1)}{n(n-1)}. \end{aligned}$$

An even simpler application of the Hook Length Formula shows that

$$f_\lambda = \binom{n}{\lambda_1,\, \lambda_2,\, n-\lambda_1-\lambda_2} \cdot \frac{\lambda_2\,(\lambda_1-\lambda_2+1)}{(n-\lambda_1)\,(n-\lambda_2+1)},$$

as required.

Chapter 5

Submodular functions

The submodular inequality, introduced in Eq. (3.1), is a very natural structural property of a function. Functions which satisfy this inequality are known as ***submodular functions***. As mentioned in Chapter 1, submodular functions arise naturally in many areas, particularly in combinatorial optimization.

In this chapter, we discuss two problems relating to submodular functions. The first is submodular function minimization (SFM). It is known that this problem can be solved using only a polynomial number of queries to the function. In Section 5.2, we prove lower bounds on the number of queries needed to solve SFM. We give three distinct proofs that, given a submodular function $f : 2^E \to \mathbb{R}$, at least $n = |E|$ queries are needed to find a minimizer. We also show that finding *all* minimizers requires at least $\Omega(n^2/\log n)$ queries.

In Section 5.3, we consider the question of "learning" (or "approximating") a given submodular function $f : 2^E \to \mathbb{R}$. Let $n = |E|$. Can we make only $\mathrm{poly}(n)$ queries to f, then approximate the value of f on *every* subset of E? We show that any such algorithm must have (worst-case) approximation ratio $\Omega(\sqrt{n/\log n})$.

5.1 Preliminaries

In this section, we state some of the key definitions and results concerning submodular functions. Further information may be found in Lovász's survey [59], McCormick's survey [63], Fujishige's monograph [30], or Schrijver's book [86].

Let E be a finite set and let f be a function that assigns a real value to each

subset of E, i.e., $f : 2^E \to \mathbb{R}$. We say that f is ***submodular*** if, for every two sets $U, V \subseteq E$, we have

$$f(U) + f(V) \geq f(U \cup V) + f(U \cap V). \tag{5.1}$$

We say that f is ***non-decreasing*** (or ***monotone***) if $f(A) \leq f(B)$ whenever $A \subseteq B$. We say that f is ***non-negative*** if $f(A) \geq 0$ for all $A \subseteq E$.

Example 5.1 (Graph cut functions). One prototypical example of a submodular function comes from graph theory. Let $G = (V, E)$ be an undirected graph. For $A \subseteq V$, let $\delta(A)$ denote the set of edges with one endpoint in A and the other in $\overline{A}$. Define $f : 2^V \to \mathbb{N}$ to be the ***cut function***: $f(A) = |\delta(A)|$. To see that f is submodular, one may verify that edges with one endpoint in $A \setminus B$ and the other in $B \setminus A$ contribute to $f(A) + f(B)$ but not to $f(A \cup B) + f(A \cap B)$; all other edges contribute equally to both.

A similar example arises from a directed graph $G = (V, A)$. Similar reasoning shows that the function $|\delta^+(\cdot)| : 2^V \to \mathbb{R}_{\geq 0}$ is a submodular function, where $\delta^+(U)$ denotes the set of arcs with their tail in U and head in $\overline{U}$. ■

The following basic facts are easy to verify.

Fact 5.2. *If f and g are submodular functions then $f + g$ is submodular.*

Fact 5.3. *Suppose that $f : 2^E \to \mathbb{R}$ is submodular. Then $g(A) := f(E \setminus A)$ is also submodular. Moreover, if f is non-decreasing then g is non-increasing (and vice-versa).*

5.1.1 Minimization

Submodular functions are, in several ways, analogous to convex functions. One similarity is that a minimizer of a submodular function can be efficiently computed. Formally, the ***submodular function minimization*** (SFM) problem is:

$$\min_{F \subseteq E} f(F) \qquad \text{where } f : 2^E \to \mathbb{R} \text{ is submodular.} \tag{5.2}$$

In order to discuss the computational efficiency of solving this problem, we must specify the computational model. As was the case with matroids, one difficulty is that submodular functions may, in general, require exponential space to

represent. Thus, we will again use an ***oracle model*** of computation. The most common such oracle is a ***value oracle***, which answers the following queries: given a set $A \subseteq S$, what is $f(A)$?

Grötschel, Lovász, and Schrijver showed that the ellipsoid method can be used to solve SFM in polynomial time [39], and even in strongly polynomial time [40]. According to McCormick [63, Theorem 2.7], this strongly polynomial algorithm has running time $\tilde{O}(n^7)$ and uses $\tilde{O}(n^5)$ oracle queries. Cunningham [18] posed the question of finding a *combinatorial*, strongly polynomial time algorithm to solve SFM; here, a "combinatorial" algorithm means one which does not rely on the ellipsoid method. This question was resolved by Iwata, Fleischer and Fujishige [47], and by Schrijver [85]. The fastest known algorithm is due to Orlin [74]; it has running time $O(n^6)$ and uses $O(n^5)$ oracle queries.

Why would one be interested in minimizing a submodular function? It turns out that many combinatorial optimization problems are special cases of SFM. For example, matroid intersection is a special case. Let $\mathrm{M}_1 = (S, r_1)$ and $\mathrm{M} = (S, r_2)$ be matroids. As stated in Eq. (3.1), both r_i are submodular. Thus, using Fact 5.2 and Fact 5.3, the function $f : 2^S \to \mathbb{Z}$ defined by $f(A) = r_1(A) + r_2(S \setminus A)$ is also submodular. Edmonds' min-max relation (Fact 3.1) shows that the maximum cardinality of an intersection of M_1 and M_2 can be found by minimizing f.

One interesting fact concerning submodular functions is that their minimizers form a ***ring family*** (sublattice of the Boolean lattice). This is analogous to the simple fact that the minimizers of a convex function form a convex set.

Fact 5.4. *Let $f : 2^E \to \mathbb{R}$ be submodular. Then the collection of minimizers of f is closed under unions and intersections.*

5.2 Lower bounds for minimization

As mentioned above, a submodular function $f : 2^E \to \mathbb{R}$ can be minimized using only $O(n^5)$ oracle queries, where $n = |E|$. It has been an outstanding open question to prove a lower bound on the number of queries needed (see the surveys of Iwata [46] or McCormick [63, p387]). Our lower bound for matroid intersection given in Chapter 4 resulted from an attempt to prove strong lower bounds for submodular function minimization. In this section, we give several proofs that $\Omega(n)$ queries are needed.

5.2.1 Linear functions

We begin by giving an adversary argument that at least n queries are needed even to minimize a *linear* function. (Linear functions are trivially submodular.) Suppose that the algorithm performs $q < n$ queries. The adversary's behavior is very simple: it returns 0 to each of those queries.

We now argue that the algorithm cannot yet have decided the minimum value of the function, since there exist two functions f and g, both consistent with the current queries, such that the minimum of f is 0, and the minimum of g is strictly negative. The existence of f is trivial: it is the zero function. To argue the existence of g requires more work.

The function $g : 2^E \to \mathbb{R}$ will have the form $g(S) = \chi(S)^\mathsf{T} a = \sum_{s \in S} a_s$, for some vector $a \in \mathbb{R}^E$. Let the sets queried by the algorithm be $S_1, S_2, \ldots, S_q$. The responses to the queries require that $\chi(S_i)^\mathsf{T} a = 0$, for all i. In other words, the vector a is required to be orthogonal to q specific vectors. Since $q < n$, there exists a subspace H (of dimension at least 1) such that all vectors $a \in H$ satisfy the required orthogonality conditions. In particular, we may choose a such that $a_j < 0$ for some j, and hence the minimum of g is strictly negative.

Suppose that we restrict our attention to integer-valued functions. How large do the values of g need to be? Let X be the matrix whose i^{th} row is the vector $\chi(S_i)$. Then we seek a non-zero, integer solution to the linear system $Xa = 0$. Let us define a new matrix $\tilde{X}$ by choosing a linearly independent set of $r := \operatorname{rank} X$ rows from X, then adding additional $n - r$ rows such that $\tilde{X}$ is non-singular, and the new rows have entries in $\{0, 1\}$. Let $b \in \mathbb{R}^E$ have its first r entries equal to zero, and last $n - r$ entries equal to -1. The vector $a := \tilde{X}^{-1} b$ satisfies the desired orthogonality conditions, but it is not necessarily integer-valued. Each entry of $\tilde{X}^{-1}$ is of the form

$$(\tilde{X}^{-1})_{i,j} \;=\; \pm \det \tilde{X}_{\mathrm{del}(j,i)} \;/\; \det \tilde{X},$$

by Eq. (2.1), and hence $(\det \tilde{X}) \cdot a$ is integer-valued. We may bound $\det \tilde{X}$ as follows.

Fact 5.5. *Let M be an $n \times n$ real matrix with entries in $\{0, 1\}$. Then $|\det M| \leq n^{n/2}$.*

Proof. Since the entries of M are in $\{0, 1\}$, we have $\|M_{*,i}\| \leq \sqrt{n}$. Hadamard's inequality [40, Eq. 0.1.28] states that $|\det M| \leq \prod_{i=1}^{n} \|M_{*,i}\|$, so the result follows. ■

Thus $|\det \tilde{X}_{\mathrm{del}(j,i)}| \leq n^{n/2}$, and thus each entry of $(\det \tilde{X}) \cdot a$ has absolute value

at most $n^{n/2+1}$. The desired integer-valued function g is obtained by setting

$$g(S) \;=\; (\det \tilde{X}) \cdot \chi(S)^{\mathsf{T}} a.$$

We summarize this discussion with the following theorem.

Theorem 5.6. *Any deterministic algorithm which decides whether an integer-valued, linear function $f : 2^E \to [-n^{O(n)}, n^{O(n)}]$ has minimum value zero requires at least n queries.*

5.2.2 Intersection of rank-1 matroids

In this section, we show a matching lower bound for submodular functions which take a significantly smaller range of values.

Theorem 5.7. *Let E be a ground set with $|E| = n$. Any deterministic algorithm which finds a minimizer of an integer-valued, submodular function $f : 2^E \to \{0, 1, 2\}$ requires at least n queries.*

Recall from Section 5.1.1 that the matroid intersection problem is a special case of submodular function minimization. We will construct a family of hard instances for SFM through the use of matroids. Let $X \subseteq S$ be arbitrary. Let $\mathrm{M}(X)$ and $\mathrm{M}(\overline{X})$ be rank-1 matroids, using the notation of Section 4.1.1. The corresponding rank functions are

$$r_{\mathrm{M}(X)}(\overline{A}) \;=\; \begin{cases} 1 & \text{if } X \cap \overline{A} \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad r_{\mathrm{M}(\overline{X})}(A) \;=\; \begin{cases} 1 & \text{if } \overline{X} \cap A \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Following Edmonds' min-max relation, let $f_X : 2^S \to \mathbb{Z}$ be defined by

$$f_X(A) \;=\; r_{\mathrm{M}(X)}(\overline{A}) + r_{\mathrm{M}(\overline{X})}(A).$$

Then f_X is submodular and $f_X(A) \in \{0, 1, 2\}$ for all $A \subseteq S$. Furthermore, $f_X(A) = 0$ if $A = X$ and f_X is strictly positive on all other points.

Let $\mathcal{F} = \{\, f_X \,:\, X \subseteq S \,\}$. Consider any algorithm which, given $f \in \mathcal{F}$, attempts to construct the (unique) minimizer of f using at most $n - 1$ queries. Without loss of generality, we may assume that the algorithm has the following behavior: if it

receives a query response of 0, it immediately halts and announces the solution. Therefore all query responses except the last one are in $\{1, 2\}$, and the last query response is in $\{0, 1, 2\}$. It follows that the number of different response sequences is at most $2^{n-2} \cdot 3 < 2^n = |\mathcal{F}|$. So, by the pigeonhole principle, two distinct functions $f_X, f_Y \in \mathcal{F}$ have the same response sequences and are therefore indistinguishable to the algorithm. Since f_X and f_Y have distinct minimizers, it follows that the algorithm cannot correctly identify the minimizer for at least one of those functions. This completes the proof of Theorem 5.7.

5.2.3 Graph cut functions

Let $G = (V, E)$ be an undirected graph and let $f : 2^V \to \mathbb{N}$ be the cut function, i.e., $f(U)$ equals the number of edges with one endpoint in U and the other in $\overline{U}$. This is a ***symmetric*** function, meaning that $f(U) = f(V \setminus U)$ for all $U \subseteq V$. Finding a minimizer of a symmetric, submodular function is trivial.

Fact 5.8. *If $f : 2^V \to \mathbb{R}$ is symmetric and submodular then $\emptyset$ and V are minimizers.*

Proof. If A is a minimizer then symmetry implies that $V \setminus A$ is too. Their intersection and union are both minimizers since minimizers form a ring family, by Fact 5.4. This proves the claim. ■

Thus, for a symmetric, submodular function, the interesting problem is to find a "non-trivial" minimizer: a set $\emptyset \neq A \subsetneq V$ such that $A \in \arg\min f$. We now show that communication complexity can be used to give a lower bound for this problem, by building on a result of Hajnal-Maass-Turán [41] and Lovász-Saks [62]. The requisite definitions are in Section 4.2.1.

Theorem 5.9. *Let $|V| = n$. Any algorithm that constructs a non-trivial minimizer of a symmetric, submodular function $f : 2^V \to \mathbb{R}$ requires $\Omega(n)$ queries, even if f is a graph cut function.*

Proof. Let G_A and G_B be graphs with vertex set V such that every possible edge $\{u, v\}$ with $u, v \in V$ is in exactly one of G_A and G_B. (So we have partitioned the complete graph into subgraphs G_A and G_B.) The edges in G_A are "owned" by Alice and those in G_B are owned by Bob. Alice is given a subgraph $H_A = (V, E_A)$ of G_A and Bob is given a subgraph $H_B = (V, E_B)$ of G_B. The problem CONN is for Alice and Bob to decide whether $H = (V, E_A \cup E_B)$ is connected. Hajnal et al. [41]

prove that the deterministic communication complexity $D(\text{CONN}) = \Omega(n \log n)$.

This yields a lower bound on the number of queries performed by any algorithm that finds a non-trivial minimizer of a graph cut function. Let

$$f(U) = |\delta_H(U)| = |\delta_{H_A}(U)| + |\delta_{H_B}(U)|.$$

Any algorithm that finds a non-trivial minimizer of f using q queries can be transformed into a communication protocol for CONN using $4q \log n$ bits of communication. This protocol works as follows. Alice and Bob both independently simulate the given algorithm. Every time the algorithm performs a query $f(U)$, Alice transmits $|\delta_{H_A}(U)|$ to Bob, and Bob transmits $|\delta_{H_B}(U)|$ to Alice. Since H_A and H_B each have less than n^2 edges, each transmission requires less than $2 \log n$ bits.

The lower bound for $D(\text{CONN})$ implies that $q = \Omega(n)$. ■

The following theorem shows an upper bound for deciding graph connectivity.

Theorem 5.10. *Let $|V| = n$ and let $f : 2^V \to \mathbb{R}$ be the cut function corresponding to graph $G = (V, E)$. There exists an algorithm using $O(n \log n)$ queries to f which can decide whether G is connected.*

Proof. The algorithm proceeds as follows. We maintain a set $U \subseteq V$ on which we have found a spanning tree. Initially U contains a single vertex.

A general step of the algorithm seeks to find an edge $\{u, v\} \in E[U, \overline{U}]$ by binary search. First we compute $|E[U, \overline{U}]| = f(U)$. If this value is zero, then G is disconnected and the algorithm halts. Otherwise, we partition $U = U_1 \dot\cup U_2$ where $|U_1| = \lfloor |U|/2 \rfloor$ and $|U_2| = \lceil |U|/2 \rceil$, then compute $|E[U_1, \overline{U}]|$ and $|E[U_2, \overline{U}]|$. This is done as follows: for any $I, J \subseteq V$ with $I \cap J = \emptyset$, we have

$$|E[I, J]| = \big(f(I) + f(J) - f(I \cup J)\big)/2. \tag{5.3}$$

So $|E[U_1, \overline{U}]|$ and $|E[U_2, \overline{U}]|$ can each be computed with three queries to f. At least one of these two values must be non-zero, say $|E[U_1, \overline{U}]| > 0$. The algorithm continues partitioning U_1 until it finds a vertex $u \in U$ such that $|E[\{u\}, \overline{U}]| > 0$. Then, the set $\overline{U}$ is similarly partitioned to find $v \in \overline{U}$ such that $|E[\{u\}, \{v\}]| > 0$, i.e., $\{u, v\}$ is an edge. Thus $O(\log n)$ queries suffice to find the edge $\{u, v\}$.

The algorithm then augments $U := U + v$ and repeats the above step. The algorithm halts when $U = V$. Thus, the total number of queries is $O(n \log n)$. ■

5.2.4 Finding all minimizers

Let $f : 2^S \to \mathbb{R}$ be a submodular function. As shown in Fact 5.4, the collection $\arg\min f$ is a ring family. It is known [6] that any ring family can be represented compactly, as follows. First, note that there is a minimal minimizer $T_{\min}$ (this follows from Fact 5.4). There exists a directed graph $G = (V, A)$ and a collection of pairwise-disjoint subsets $T_v \subseteq S \; \forall v \in V$ such that

$$\arg\min f \;=\; \left\{ T_{\min} \cup \bigcup_{v \in U} T_v \;:\; U \text{ is a sink set for } G \right\}.$$

A ***sink set*** is a subset of the vertices which has no leaving arcs.

Given a submodular function f, this compact representation of $\arg\min f$ can be explicitly constructed using only $O(n^5)$ oracle queries to f, via Orlin's algorithm [74]; see Murota [70, p290] for details. The following theorem shows a superlinear lower bound on the number of queries needed to solve this problem.

Theorem 5.11. *Any deterministic algorithm which constructs* $\arg\min f$ *(or any compact representation thereof) must perform* $\Omega(n^2/\log n)$ *queries.*

Suppose that n is even and let U and V be disjoint sets of vertices with $|U| = |V| = n/2$. Let $S = U \dot\cup V$. Let $\mathcal{G}$ be the family of all directed graphs on vertex set S for which all arcs have their tail in U and head in V. Clearly $|\mathcal{G}| = 2^{n^2/4}$.

Claim 5.12. *If* $G, H \in \mathcal{G}$ *are distinct then there exists a set* $T \subseteq S$ *such that* T *is a sink set for* G *but not for* H *(or vice-versa).*

Proof. Without loss of generality, there exists an arc $e = (u, v)$ of G that is not an arc of H, with $u \in U$ and $v \in V$. Let A_H be the set of all arcs in H. Let

$$T \;=\; \{\, w \;:\; (u, w) \in A_H \,\} \;+\; u,$$

so $v \notin T$. Then T is a sink set for H but not a sink set for G. ∎

For any digraph $G \in \mathcal{G}$, let $f_G : 2^S \to \mathbb{N}$ be the cut function, namely $f_G(U) = |\delta^+_G(U)|$. As argued in Example 5.1, f_G is submodular. The minimizers of f_G are precisely the sink sets, since f_G is non-negative. So Claim 5.12 shows that, if $G, H \in \mathcal{G}$ are distinct, then $\arg\min f_G \neq \arg\min f_H$.

Let $\mathcal{F} = \{ f_G : G \in \mathcal{G} \}$. Consider any algorithm which, given $f \in \mathcal{F}$, attempts to construct $\arg\min f$ using fewer than $n^2/(8 \log n)$ queries. Each query response is an integer between 0 and $n^2/4$. So the number of different response sequences is less than

$$\left(n^2\right)^{n^2/(8 \log n)} = 2^{n^2/4} = |\mathcal{G}| = |\mathcal{F}|.$$

So, by the pigeonhole principle, two distinct functions $f_G, f_H \in \mathcal{F}$ have the same response sequences and are therefore indistinguishable to the algorithm. As argued above, $\arg\min f_G \neq \arg\min f_H$, and therefore the algorithm cannot correctly compute the set of minimizers for at least one function in $\mathcal{F}$. This completes the proof of Theorem 5.11.

5.3 Learning a submodular function

In this section, we investigate another fundamental question concerning the structure of submodular functions. How much information is contained in a submodular function? How much of that information can be learned in just a few queries? To address these questions, we consider a framework in which an algorithm attempts to "learn" a submodular function approximately using only a polynomial number of queries. Formally, we consider the following problem.

Problem 5.13. *Let $f : 2^E \to \mathbb{R}_{\geq 0}$ be a submodular function with $n = |E|$. Can one make $n^{O(1)}$ queries to f and construct a function $\hat{f}$ which is an approximation of f, in the sense that $\hat{f}(S) \leq f(S) \leq g(n) \cdot \hat{f}(S)$ for all $S \subseteq E$. The quantity $g(n)$ is called the **approximation ratio**. What approximation ratios can be obtained? Can one obtain a constant approximation ratio (independent of n)?*

This problem can be solved exactly (i.e., with approximation ratio 1) for certain classes of submodular functions. As a simple example, Lemma 5.14 shows this for graph cut functions. In contrast, Section 5.3.1 shows that, for general submodular functions, one needs to take $g(n) = \Omega(\sqrt{n/\log n})$. Even when restricting to monotone functions, we show that $g(n) = \Omega(\sqrt{n}/\log n)$. By a slightly different choice of parameters, Svitkina and Fleischer [89] have improved our analysis to show that $g(n) = \Omega(\sqrt{n/\log n})$ for monotone functions.

Lemma 5.14. *Let $G = (V, E)$ be a graph and let $f : 2^V \to \mathbb{N}$ be the cut function, i.e.,*

$f(U) = |\delta(U)|$. Given f, the graph G can be constructed using $O(|V|^2)$ queries.

Proof. Let $I, J \subseteq V$ satisfy $I \cap J = \emptyset$. We showed earlier in Eq. (5.3) that $|E[I, J]|$ can be computed using only three queries to f. This leads to the following procedure for reconstructing G given f. For each pair of vertices u and v, we compute $|E[\{u\}, \{v\}]|$ using Eq. (5.3). If this value is 1 then $\{u, v\} \in E$, otherwise $\{u, v\} \notin E$. Thus $O(|V|^2)$ queries suffice. ■

5.3.1 Lower bound via laminar matroids

The bulk of this section shows that Problem 5.13 requires an approximation ratio of $\Omega(\sqrt{n}/\log n)$, even when restricting f to be a matroid rank function (and hence a monotone, submodular function). At the end of the section, we discuss the slightly stronger result obtained without the monotonicity assumption.

Our construction depends on two parameters $0 < \beta < \alpha < n$, to be specified later. Let U be the uniform rank-α matroid on E. We will construct another matroid M which cannot easily be distinguished from U. Let r_{U} and r_{M} be these matroids' respective rank functions. Let R be a fixed set of cardinality α, and note that $r_{\mathrm{U}}(R) = \alpha$.

We now construct M_R such that $r_{\mathrm{M}_R}(R) = \beta$. Define the laminar family $\mathcal{L} = \{R, E\}$, with $d_R = \beta$ and $d_E = \alpha$. Let $\mathrm{M}_R = (E, \mathcal{I}_{\mathcal{L},d})$, be the corresponding laminar matroid. (Recall the definition from page 35.) When we do not wish to emphasize the parameter R, we denote this matroid simply by $\mathrm{M} = (E, \mathcal{I}_{\mathrm{M}})$.

Consider any algorithm for approximating a monotone submodular function. We will give it as input either $f = r_{\mathrm{U}}$ or $f = r_{\mathrm{M}}$, for some choice of R. If the algorithm cannot distinguish between these two possibilities, then it cannot approximate f better than

$$\frac{r_{\mathrm{U}}(R)}{r_{\mathrm{M}_R}(R)} = \frac{\alpha}{\beta}. \tag{5.4}$$

Our objective is to maximize this ratio, while ensuring the algorithm cannot distinguish between $f = r_{\mathrm{U}}$ or $f = r_{\mathrm{M}}$.

Suppose that the algorithm's first step is to query a set S. If $f(S) \neq r_{\mathrm{U}}(S)$ then the algorithm knows that $f = r_{\mathrm{M}_R}$ for some R, and we will pessimistically assume that it can also identify R at this point. Otherwise, suppose that $f(S) = r_{\mathrm{U}}(S)$. Then the algorithm knows that either $f = r_{\mathrm{U}}$, or that $f = r_{\mathrm{M}_R}$ but there are now

some restrictions on what R can be. Let us now explore these restrictions on R. For convenience, let $s = |S|$.

Case 0: $s \leq \beta$. In this case, we have $f(S) = r_{\mathbf{U}}(S) = r_{\mathbf{M}_R}(S)$ for all R, so the algorithm learns nothing.

Case 1: $\beta < s \leq \alpha$. In this case, $r_{\mathbf{U}}(S) \neq r_{\mathbf{M}}(S)$ iff $S \notin \mathcal{I}_{\mathbf{M}}$, which holds iff $|R \cap S| > \beta$. So if $r_{\mathbf{U}}(S) = r_{\mathbf{M}}(S)$ then the algorithm has learned that R cannot be any of the sets of size α that intersect S in more than β elements. We will choose α and β such that an $n^{-\omega(1)}$ fraction of possible sets R is eliminated.

The analysis proceeds using the language of probability. We wish to analyze the probability $\Pr[\,|R \cap S| > \beta\,]$, where R is chosen uniformly at random among all sets of size α. This is precisely $\Pr[\,|R \cap S| > \beta \mid |R| = \alpha\,]$, where R is now chosen by picking each element independently with probability α/n. Now we use the standard trick

$$\begin{aligned}\Pr[\,|R \cap S| > \beta \mid |R| = \alpha\,] &= \frac{\Pr[\,|R \cap S| > \beta \wedge |R| = \alpha\,]}{\Pr[\,|R| = \alpha\,]} \\ &\leq (n+1) \cdot \Pr[\,|R \cap S| > \beta\,].\end{aligned}$$

where the last inequality follows since the binomial distribution with parameter α/n has its mode equal to α.

The probability $\Pr[\,|R \cap S| > \beta\,]$ (where elements are chosen independently) can easily be analyzed using Chernoff bounds. Let $\mu = \mathrm{E}[\,|R \cap S|\,] = s\alpha/n \leq \alpha^2/n$. Let us take $\alpha = \sqrt{n}$, so that $\mu \leq 1$. Then taking $\beta = \omega(\log n)$, independent of s, implies that $\Pr[\,|R \cap S| > \beta\,] \leq n^{-\omega(1)}$.

To summarize Case 1, if we take $\alpha = \sqrt{n}$ and $\beta = \omega(\log n)$, the number of possible sets R eliminated by a query is a $n^{-\omega(1)}$ fraction of the possible sets R.

Case 2: $s > \alpha$. Since $r_{\mathbf{M}}(S) \leq r_{\mathbf{U}}(S) = \alpha$ for all such sets, we wish to understand when $r_{\mathbf{M}}(S) < \alpha$. Note that $S \notin \mathcal{I}_{\mathbf{M}}$ (it is too big) and that $r_{\mathbf{M}}(S) = \max_{I \subset S, I \in \mathcal{I}_{\mathbf{M}}} |I|$. Let $t = |R \cap S| - \beta$. If $t \leq 0$ then any α-element subset I of S is in $\mathcal{I}_{\mathbf{M}}$, so $r_{\mathbf{M}}(S) = \alpha$. Otherwise, suppose we modify S by eliminating t elements from $R \cap S$; call the resulting set J. The resulting set might not be independent (it may have cardinality bigger than α). In any case, we have $r_{\mathbf{M}}(S) = \alpha$ iff $|J| \geq \alpha$. Since $|J| = s - |R \cap S| + \beta$,

we have argued that $r_{\mathbf{M}}(S) = r_{\mathbf{U}}(S)$ iff $|R \cap S| \leq s - \alpha + \beta$. Since $|R| = \alpha$, this condition is always satisfied if $s \geq 2\alpha - \beta$. So we may assume that $\alpha < s \leq 2\alpha - \beta$.

The analysis is now similar to the analysis of Case 1. We wish to bound the probability $\Pr[\,|R \cap S| > s - \alpha + \beta\,]$, where R is picked uniformly at random from sets of size α. This is upper bounded by $(n+1)\cdot\Pr[\,|R \cap S| > s - \alpha + \beta\,]$, where elements of R are picked independently with probability α/n. Now $\mu = \mathrm{E}[\,|R \cap S|\,] = s\alpha/n \leq 2\alpha^2/n$. As above, we may take $\beta = \omega(\log n)$, and conclude that a query eliminates a $n^{-\omega(1)}$ fraction of the possible sets R.

Summary. After $n^{O(1)}$ queries, the total fraction of possible sets R that have been eliminated is

$$n^{O(1)} \cdot n^{-\omega(1)} = n^{-\omega(1)}.$$

Since at least one possible set R remains, the algorithm cannot have distinguished between $f = r_{\mathbf{U}}$ and $f = r_{\mathbf{M}}$. By Eq. (5.4), we conclude that no adaptive algorithm can approximate a monotone submodular function to within $\Omega\big(\sqrt{n}/\log n\big)$.

A non-monotone variant. The previous construction can be modified to obtain an interesting non-monotone submodular function. The main idea is to "tilt" f downwards by subtracting a linear function. This allows us to choose α and β slightly differently. Formally, we define f according to one of the following definitions.

$$\begin{aligned} \textit{Definition A:} \quad & f(S) = r_{\mathbf{U}}(S) - |S|/2 \\ \textit{Definition B:} \quad & f(S) = r_{\mathbf{M}_R}(S) - |S|/2 \end{aligned}$$

We will now set $\alpha = n/2$ and $\beta = n/4 + \omega(\sqrt{n \log n})$. Under Definition A, we have $f(R) = n/4$, but under Definition B we have $f(R) \approx \sqrt{n \log n}$. By similar probabilistic arguments, one can show that no algorithm using only polynomially many queries can distinguish between Definitions A and B. In summary, no algorithm can approximate a non-monotone submodular function to within $\Omega\big(\sqrt{n/\log n}\big)$. This gives a slightly stronger lower bound than the previous result, although the previous result holds even for monotone functions.

5.4 Discussion

Matroid intersection. As discussed in Section 5.1.1 and Section 5.2.2, matroid intersection is a special case of SFM. Thus query lower bounds for matroid intersection can lead to query lower bounds for submodular function minimization as well, depending on the precise oracle models used. Specifically, Edmonds' min-max relation and a standard simulation argument shows that $D(\text{MATINT})/2 \log r$ gives a lower bound on the number of queries needed to minimize a submodular function. This may be a fruitful approach to proving lower bounds for SFM.

Graph connectivity. Section 5.2.3 considers the problem of finding a non-trivial minimizer of a graph cut function, i.e., deciding whether the corresponding graph is connected. It shows that $\Omega(n)$ queries are necessary and $O(n \log n)$ queries are sufficient. What is the exact number of queries needed?

Learning. Can one show an upper bound of $g(n) = \tilde{O}(\sqrt{n})$ for learning a submodular function? Svitkina and Fleischer [89] solve this problem for for a class of monotone submodular functions which contains the monotone functions used in our lower bound. Can such arguments be extended to a broader class of submodular functions?

Appendix A

Fast triangular factorization

In this chapter, we give an algorithm to factorize a $n \times n$ matrix A into lower- and upper-triangular matrices in $O(n^\omega)$ time. This algorithm applies even if A is singular. It can also be used to compute a maximum-rank submatrix of A, the determinant of A, and to invert A if it is non-singular, all in $O(n^\omega)$ time.

Such algorithms have been known for decades, so the material of this chapter should not be considered novel. However, it seems difficult to find a concise explanation of fast matrix factorization in the literature. Since Chapters 2 and 3 use these algorithms extensively, this appendix provides a self-contained discussion.

The algorithm we present here is similar in spirit to the algorithm of Bunch and Hopcroft [8] for finding the LUP decomposition of a matrix. However the algorithms are not identical — for example, the Bunch-Hopcroft algorithm requires that the given matrix is non-singular.

A.1 Overview

First, some definitions are needed. We say that a matrix D is a ***generalized permutation matrix*** if each row and each column contains at most one non-zero entry. In other words, D is the product of a diagonal matrix and a permutation matrix.

The objective is as follows. We are given an $n \times n$ matrix A. Our goal is to compute $D = L \cdot A \cdot U$, where L is lower-triangular, U is upper-triangular, and D is a generalized permutation matrix.

Intuitively, the algorithm makes progress towards making D a generalized per-

Algorithm A.1 A sketch of the recursive algorithm for triangular factorization of a matrix.

- Set $D^{(1)} := A$ and partition $D^{(1)}$ into four submatrices $D^{(1)} = \begin{pmatrix} W^{(1)} & X^{(1)} \\ Y^{(1)} & Z^{(1)} \end{pmatrix}$.
- Recursively factor $W^{(1)}$, obtaining $d^{(1)} = l^{(1)} \cdot W^{(1)} \cdot u^{(1)}$.
- Use this to get factorization $D^{(2)} = L^{(2)} \cdot A \cdot U^{(2)}$, where $D^{(2)} = \begin{pmatrix} W^{(2)} & X^{(2)} \\ Y^{(2)} & Z^{(2)} \end{pmatrix}$, and $W^{(2)}$ is a generalized permutation matrix.
- Recursively factor $X^{(2)}$, obtaining $d^{(2)} = l^{(2)} \cdot X^{(2)} \cdot u^{(2)}$.
- Use this to get factorization $D^{(3)} = L^{(3)} \cdot A \cdot U^{(3)}$, where $D^{(3)} = \begin{pmatrix} W^{(3)} & X^{(3)} \\ Y^{(3)} & Z^{(3)} \end{pmatrix}$, and $W^{(3)}$ and $X^{(3)}$ are generalized permutation matrices.
- Recursively factor $Y^{(3)}$, obtaining $d^{(3)} = l^{(3)} \cdot Y^{(3)} \cdot u^{(3)}$.
- Use this to get factorization $D^{(4)} = L^{(4)} \cdot A \cdot U^{(4)}$, where $D^{(4)} = \begin{pmatrix} W^{(4)} & X^{(4)} \\ Y^{(4)} & Z^{(4)} \end{pmatrix}$, and $W^{(4)}$, $X^{(4)}$ and $Y^{(4)}$ are generalized permutation matrices.
- Recursively factor $Z^{(4)}$, obtaining $d^{(4)} = l^{(4)} \cdot Z^{(4)} \cdot u^{(4)}$.
- Use this to get the desired factorization $D = L \cdot A \cdot U$.

mutation matrix by performing ***pivot*** operations. Roughly speaking, a pivot operation on an entry $D_{i,j} \neq 0$ is a sequence of elementary row and column operations which ensure that row $D_{i,j}$ is the only non-zero entry in row i and column j. Ordinary Gaussian elimination follows essentially the same strategy, iteratively performing pivot operations one-by-one.

Our algorithm works recursively rather than iteratively. A sketch of the algorithm is shown in Algorithm A.1. The algorithm recursively factors four smaller matrices, each one half the size of A. After each recursive step, it updates the factorization and the matrix D. A concrete implementation of this algorithm is given in Algorithm A.2, and the following section explains the algorithm in detail.

A.2 Algorithm

Formally, the algorithm maintains $n \times n$ matrices L, U and D, and sets R and C of row and column indices such that the following properties hold.

P_1: $D = L \cdot A \cdot U$,

Algorithm A.2 Matlab code for the algorithm to compute a triangular factorization of a matrix. It finds L, U, D such that L is lower-triangular with unit diagonal, U is upper-triangular with unit diagonal, D is a generalized permutation matrix, and $D = L \cdot A \cdot U$.

```
%%% FactorMatrix %%%
function [L,D,U,R,C] = FactorMatrix(A)

% Base case
n = size(A,1);
if n==1
    L=[1]; D=A; U=[1];
    if abs(A(1,1))<100*eps R=[]; C=[];
    else R=[1]; C=[1]; end;
    return;
end

m = floor(n/2);
L=eye(n); U=L; D=A;
indices{1}=1:m; indices{2}=m+1:n;
R=[]; C=[];

for i=1:2
    I=indices{i}; IComp=indices{3-i};

    for j=1:2
        J=indices{j}; JComp=indices{3-j};

        [l,d,u,r,c] = FactorMatrix( D(I,J) );
        r=I(r); c=J(c); % Translate indices
        R=[R,r]; C=[C,c];

        % Extend l and u to operators L' and U'
        newL=zeros(n); newL(I,I)=l; newL(IComp,IComp)=eye(length(IComp));
        newU=zeros(n); newU(J,J)=u; newU(JComp,JComp)=eye(length(JComp));
        L=newL*L; U=U*newU; D=newL*D*newU;

        % invdrc = inverse of submatrix of D in which we pivoted
        invdrc = InvertGenPerm(D(r,c));

        % Eliminate in rows r and cols c
        newL=eye(n); newL(IComp,r)=-D(IComp,c)*invdrc;
        newU=eye(n); newU(c,JComp)=-invdrc*D(r,JComp);
        L=newL*L; U=U*newU; D=newL*D*newU;
    end
end

%%% InvertGenPerm %%%
function B = InvertGenPerm(A)
n = size(A,1);
B = zeros(n);
for i=1:n
    for j=1:n
        if abs(A(i,j))>.0001
            B(j,i) = 1/A(i,j);
        end;
    end;
end;
```

P_2: L is lower-triangular with unit-diagonal,

P_3: U is upper-triangular with unit-diagonal, and

P_4: $D_{R,C}$ is a non-singular, generalized permutation matrix,

P_5: $D_{R,\overline{C}} = 0$ and $D_{\overline{R},C} = 0$,

P_6: if $i \notin C$ then $U_{i,*} = e_i^\mathsf{T}$,

P_7: if $A_{*,j} = 0$ then $U_{*,j} = e_j$,

P_8: if $j \notin R$ then $L_{*,j} = e_j$,

P_9: if $A_{i,*} = 0$ then $L_{i,*} = e_i^\mathsf{T}$.

Let $P_i(A, L, D, U, R, C)$ denote the predicate that property i holds for those specific matrices. Let $P(A, L, D, U, R, C)$, or simply P, denote the predicate that all properties P_1-P_9 hold for those matrices.

The meaning of these properties is as follows. P_1-P_3 give the definition of our desired factorization. P_4-P_5 capture the algorithm's aim of making D more like a generalized permutation matrix. P_6 ensures that only columns on which the algorithm has pivoted (i.e., those in C) are added to other columns. P_7 ensures that no column operations are applied to columns which are already zero. P_8-P_9 are analogous to P_6-P_7.

The initial values for the algorithm's parameters are

$$L := I, \qquad D := A, \qquad U := I, \qquad R = \emptyset, \qquad C = \emptyset.$$

These values trivially satisfy P. The algorithm halts when the following property is satisfied:

P_h: $D_{\overline{R},\overline{C}} = 0$, which implies that D is a generalized permutation matrix.

Thus, when property P_h is satisfied, the desired factorization has been found. Let $H(A, L, D, U, R, C)$ be the predicate that properties P_1-P_9 and P_h are all satisfied.

We now explain the recursive steps. Let us write

$$D = \begin{pmatrix} W & X \\ Y & Z \end{pmatrix},$$

where the submatrices have size $m \times m$ and $m = n/2$. As explained in the sketch above, the algorithm recurses on a submatrix, then updates D, and so on. The

algorithm ensures that P and the following two properties hold throughout the algorithm.

P_{10}: $D_{R,C}$ contains non-zero entries only in submatrices into which the algorithm has already recursed.

P_{11}: For every submatrix $M \in \{W, X, Y, Z\}$, if the algorithm has already recursed on M then either $M_{i,*} = 0$ or $i \in R$. Similarly, either $M_{*,j} = 0$ or $j \in C$.

Let us now consider the situation where the algorithm has just recursively factored a submatrix and we wish to update the parameters accordingly. Rather than separately considering the scenarios in which submatrices W, X, Y and Z have just been factored, we will instead consider the equivalent situation where W has just been factored but the algorithm may have already factored some of X, Y and Z. It is important to point out that the algorithm *cannot* handle the scenario in which first Z is factored, followed immediately by W; as illustrated in Algorithm A.1, this scenario does not arise.

Henceforth, we do not assume that $R = C = \emptyset$ or that $L = U = I$.

The recursive factorization of W produces three $m \times m$ matrices[1] l, u and d, and sets of row and column indices r and c. Furthermore, $H(W, l, d, u, r, c)$ holds. By properties P_4, P_5 and P_h, we may write

$$d = \begin{pmatrix} d_{r,c} & 0 \\ 0 & 0 \end{pmatrix}, \tag{A.1}$$

where $d_{r,c}$ is a non-singular, generalized permutation matrix. It is not necessarily the case that $d_{r,c}$ is the north-west submatrix of d, but for notational simplicity we will make this assumption. It follows that W can be decomposed as

$$W = \begin{pmatrix} W_{r,c} & W_{r,\bar{c}} \\ W_{\bar{r},c} & W_{\bar{r},\bar{c}} \end{pmatrix}, \qquad \text{where } W_{r,c} \text{ is a full-rank, square submatrix.} \tag{A.2}$$

[1]Regrettably, we are now using lower-case letters for matrices and sets. This violates usual conventions, but does seem appropriate for the present discussion since, for example, the matrix l is analogous to L but smaller.

We now show how to use l, u, and d to make progress on factoring D. First of all, we have

$$\underbrace{\begin{pmatrix} l & 0 \\ 0 & I \end{pmatrix}}_{L'} \cdot \underbrace{\begin{pmatrix} W & X \\ Y & Z \end{pmatrix}}_{D} \cdot \underbrace{\begin{pmatrix} u & 0 \\ 0 & I \end{pmatrix}}_{U'} = \underbrace{\begin{pmatrix} d & \hat{X} \\ \hat{Y} & Z \end{pmatrix}}_{D'}, \tag{A.3}$$

where $\hat{X} = l \cdot X$ and $\hat{Y} = Y \cdot u$, for notational simplicity. We now wish to eliminate non-zeros from the rows in r and columns in c. This has already been done within submatrix d, as shown in Eq. (A.1). It remains to deal with $\hat{X}$ and $\hat{Y}$. This is done as follows:

$$\begin{aligned} &\underbrace{\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -\hat{Y}_{*,c} \cdot (d_{r,c})^{-1} & 0 & I \end{pmatrix}}_{L''} \cdot \underbrace{\begin{pmatrix} d_{r,c} & 0 & \hat{X}_{r,*} \\ 0 & 0 & \hat{X}_{\bar{r},*} \\ \hat{Y}_{*,c} & \hat{Y}_{*,\bar{c}} & Z \end{pmatrix}}_{D'} \cdot \underbrace{\begin{pmatrix} I & 0 & -(d_{r,c})^{-1} \cdot \hat{X}_{r,*} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix}}_{U''} \\ &= \underbrace{\begin{pmatrix} d_{r,c} & 0 & 0 \\ 0 & 0 & \hat{X}_{\bar{r},*} \\ 0 & \hat{Y}_{*,\bar{c}} & Z - \hat{Y}_{*,c} \cdot (d_{r,c})^{-1} \cdot \hat{X}_{r,*} \end{pmatrix}}_{D''}. \end{aligned} \tag{A.4}$$

We use Eq. (A.3) and Eq. (A.4) to update the algorithm's parameters as follows:

$$L := L'' \cdot L' \cdot L,\ U := U \cdot U' \cdot U'',\ D := L'' \cdot L' \cdot D \cdot U' \cdot U'',\ R := R \cup r,\ C := C \cup c. \tag{A.5}$$

This concludes the description of an update. As illustrated in Algorithm A.1, the algorithm then continues to recurse on the next submatrix, and halts when all four recursions are complete.

A.3 Analysis

Given a matrix of size n, the algorithm recursively factors four matrices of size $n/2$. After each recursive step, the algorithm must compute the matrices L', L'', U', and U'', and apply the update shown in Eq. (A.5). This involves inverting $d_{r,c}$ and performing only a constant number of matrix multiplications. Since $d_{r,c}$ is a generalized permutation matrix, it can be inverted in $O(n^2)$ time.

Thus, the algorithm can be analyzed as follows. Let $f(n)$ denote the time required to execute the algorithm on a matrix of size n. Then $f(n)$ satisfies the recurrence

$$f(n) = 4 \cdot f\left(\frac{n}{2}\right) + O\left(n^{\omega}\right). \tag{A.6}$$

By standard arguments [17], the solution of this recurrence is $f(n) = O(n^{\omega})$ if $\omega > 2$, and $f(n) = \tilde{O}(n^2)$ if $\omega = 2$.

A.4 Correctness

To argue correctness of the algorithm, we must show that

- $P(A, L, D, U, R, C)$ continues to hold after the update in Eq. (A.5),
- P_{10} and P_{11} also hold after each update, and
- $H(A, L, D, U, R, C)$ holds after the algorithm recurses on all submatrices.

P_1 follows directly from the new definition of L, U, and D. P_2 and P_3 follow from the definition of L', L'', U' and U'', and the fact that products of lower- (resp., upper-) triangular matrices are lower- (resp., upper-) triangular.

To prove the remaining properties, some technical claims are needed.

Claim A.1. *$R \cap r = \emptyset$ and $C \cap c = \emptyset$.*

Proof. Let $i \in R$. Consider the situation before recursing on submatrix W. P_5 and P_{10} imply that $W_{i,*} = 0$. However, $W_{r,c}$ is non-singular by Eq. (A.2), and thus $i \notin r$. Hence $R \cap r = \emptyset$. Similarly, $C \cap c = \emptyset$. ■

The following claim establishes the intuitive fact that rows on which the algorithm pivots are unaffected by subsequent pivots. Let L, D, U, R and C now denote their values before applying Eq. (A.5), and let D'' denote the updated value of matrix D.

Claim A.2. *Let $R' = R \cup r$ and $C' = C \cup c$. Then*

- *$D''_{R',C'}$ is a non-singular, generalized permutation matrix,*
- *$D''_{R',\overline{C'}} = 0$ and $D''_{\overline{R'},C'} = 0$.*

Proof. By Claim A.1, $R \cap r = \emptyset$ and $C \cap c = \emptyset$. We wish to show that

$$D'' = \begin{array}{c} \\ r \\ R \\ \overline{R'} \end{array} \begin{array}{c} \begin{array}{ccc} c & C & \overline{C'} \end{array} \\ \begin{pmatrix} d_{r,c} & 0 & 0 \\ 0 & D_{R,C} & 0 \\ 0 & 0 & ? \end{pmatrix} \end{array} . \tag{A.7}$$

(The submatrix $D''_{\overline{R'},\overline{C'}}$ is not relevant for this claim.)

From Eq. (A.4), it is clear that

$$D''_{r,c} = d_{r,c}, \qquad D''_{r,\overline{c}} = 0, \qquad \text{and} \qquad D''_{\overline{r},c} = 0. \tag{A.8}$$

To complete the proof, we now argue that $D''_{R,*} = D_{R,*}$. To prove this, let $i \in R$ be arbitrary. By P_4 and P_5, there exists exactly one non-zero entry in $D_{i,*}$, say $D_{i,k} \neq 0$ where $k \in C$. Thus $k \in \overline{c}$ and $i \in \overline{r}$. We wish to show that $D''_{i,j} = D_{i,j}$, where j is an arbitrary column. There are four cases to consider.

Case 1: Entry $D_{i,j}$ is in submatrix W. Since $i \in R$, P_4, P_5 and P_{10} imply that $W_{i,*} = 0$. Thus $k \neq j$, and $D_{i,j} = 0$. If $j \in c$ then Eq. (A.8) shows that $D''_{i,j} = 0$ as required. Since entry (i,j) is in the north-west submatrix but $i \in \overline{r}$, Eq. (A.4) shows that $D''_{i,j} = 0$.

Case 2: Entry $D_{i,j}$ is in submatrix X. As observed above, $W_{i,*} = 0$. By property $P_9(W,l,d,u,r,c)$, we have $l_{i,*} = e_i^\mathsf{T}$. Thus $D''_{i,j} = \hat{X}_{i,j} = l_{i,*} \cdot X_{*,j} = X_{i,j} = D_{i,j}$.

Case 3: Entry $D_{i,j}$ is in submatrix Y. Then $D''_{i,j} = \hat{Y}_{i,j} = Y_{i,*} \cdot u_{*,j}$. If k is not a column of Y then $j \neq k$ and $Y_{i,*} = 0$, so $D''_{i,j} = 0 = D_{i,j}$ as required. Otherwise, $D''_{i,j} = Y_{i,k}\, u_{k,j}$. Since $k \notin c$, $P_6(W,l,d,u,r,c)$ implies that $u_{k,*} = e_k^\mathsf{T}$. Thus, if $j = k$ we have $D''_{i,j} = Y_{i,j} = D_{i,j}$, and if $j \neq k$ we have $D''_{i,j} = 0 = D_{i,j}$, as required.

Case 4: Entry $D_{i,j}$ is in submatrix Z. Note that $\hat{Y}_{i,c} = Y_{i,*}\, u_{*,c} = Y_{i,c}\, u_{c,c} + Y_{i,\overline{c}}\, u_{\overline{c},c}$. Since $k \in \overline{c}$, $Y_{i,c} = 0$. By $P_6(W,l,d,u,r,c)$, $u_{\overline{c},c} = 0$. Thus $\hat{Y}_{i,c} = 0$, and

$$D''_{i,j} = Z_{i,j} - \hat{Y}_{i,c} \cdot (d_{r,c})^{-1} \cdot \hat{X}_{r,j} = Z_{i,j} = D_{i,j}.$$

This concludes the argument that $D''_{R,*} = D_{R,*}$. By $P_5(A,L,D,U,R,C)$, it follows that $D''_{R,\overline{C}} = 0$, as required. A symmetric argument shows that $D''_{*,C} = D_{*,C}$, and hence that $D''_{\overline{R},C} = 0$. This establishes the desired block-decomposition of

Eq. (A.7).

Now by $P_4(W, l, d, u, r, c)$, $D''_{r,c} = d_{r,c}$ is a non-singular, generalized permutation matrix. Similarly, by $P_4(A, L, D, U, R, C)$, $D''_{R,C} = D_{R,C}$ is a non-singular, generalized permutation matrix. Thus, by Eq. (A.7), it follows that $D''_{R',C'}$ is also a non-singular, generalized permutation matrix. ■

Claim A.2 directly shows that P_4 and P_5 are satisfied after the update in Eq. (A.5).

The following claim shows that P_6 holds after the update. Let $C' = C \cup c$ and let U' and U'' be as defined in Eq. (A.3) and Eq. (A.4).

Claim A.3. *If $i \notin C'$ then $(U \cdot U' \cdot U'')_{i,*} = e_i^\mathsf{T}$.*

Proof. Let $i \notin C'$ be arbitrary. By $P_6(A, L, D, U, R, C)$, we have $U_{i,*} = e_i^\mathsf{T}$. We claim that $U'_{i,*} = e_i^\mathsf{T}$. If i is not a column of W then this is immediate from the definition of U'. Otherwise, since $i \notin c$, this holds by $P_6(W, l, d, u, r, c)$. Next we claim that $U''_{i,*} = e_i^\mathsf{T}$. This is also immediate from the definition of U'', since $U''_{i,*} \neq e_i^\mathsf{T}$ only if $i \in c$. Since $U_{i,*} = e_i^\mathsf{T}$, $U'_{i,*} = e_i^\mathsf{T}$, and $U''_{i,*} = e_i^\mathsf{T}$, it follows that $(U \cdot U' \cdot U'')_{i,*} = e_i^\mathsf{T}$, as required. ■

The following claim show that P_7 holds after the update.

Claim A.4. *If $A_{*,j} = 0$ then $(U \cdot U' \cdot U'')_{*,j} = e_j^\mathsf{T}$.*

Proof. Since $A_{*,j} = 0$, $P_7(A, L, D, U, R, C)$ implies that $U_{*,j} = e_j$. Thus P_1 implies that $D_{*,j} = 0$. Next, by considering two cases, we show that $U'_{*,j} = U''_{*,j} = e_j$, which proves the claim.

First suppose that j is a column of W. Then $W_{*,j} = 0$ and, by $P_7(W, l, d, u, r, c)$, $u_{*,j} = e_j$. Thus by definition (in Eq. (A.3)), $U'_{*,j} = e_j$. Also, by definition (in Eq. (A.4)), $U''_{*,j} = e_j$.

Next suppose that j is not a column of W. Then, by definition, $U'_{*,j} = e_j$. Since $D_{*,j} = 0$, we have $X_{*,j} = 0$. Thus $(d_{r,c})^{-1} \cdot \hat{X}_{*,j} = (d_{r,c})^{-1} \cdot l \cdot X_{*,j} = 0$, and hence $U''_{*,j} = e_j$. ■

Clearly properties P_8 and P_9 are symmetric to P_6 and P_7, so they also hold after the update.

We now argue that P_{10} holds after the update. Suppose that $i \in R \cup r$ and $j \in C \cup c$ but the algorithm has not yet recursed on the submatrix containing row i and column j. We cannot have $(i, j) \in r \times c$ as the algorithm has already recursed on submatrix W. On the other hand, if $(i, j) \in (R \times c) \cup (r \times C)$ then

Eq. (A.4) shows that $D''_{i,j} = 0$ as required. Finally, if $(i,j) \in R \times C$ then, as argued in Claim A.2, $D''_{R,C} = D_{R,C}$. Thus, since P_{10} held before recursing on submatrix W, $D''_{i,j} = D_{i,j} = 0$.

Claim A.5. *Let the matrices before and after updating be, respectively,*

$$D = \begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \qquad \text{and} \qquad D'' = \begin{pmatrix} W'' & X'' \\ Y'' & Z'' \end{pmatrix}.$$

We claim that:

1. *if the algorithm has already recursed on X, then $X = X''$,*
2. *if the algorithm has already recursed on Y, then $Y = Y''$,*
3. *if the algorithm has already recursed on Z, and on either X or Y, then $Z = Z''$.*

Proof. We prove the second claim; the proof of the first claim is symmetric. We show $Y''_{*,j} = Y_{*,j}$ for any column j. If $j \in c$ then $Y''_{*,j} = 0$ by Eq. (A.4) and $j \notin C$ by Claim A.1. Thus, since P_{11} held before recursing on W, $Y_{*,j} = 0$ as required. If $j \notin c$ then $Y''_{*,j} = \sum_i Y_{*,i} u_{i,j}$. If $i \in c$ then $i \notin C$ so $Y_{*,i} = 0$ by P_{11} again. But if $i \notin c$ then $P_6(W, l, d, u, r, c)$ implies that $u_{i,j}$ is 1 if $i = j$ and 0 otherwise. Thus $Y''_{*,j} = Y_{*,j}$ as required.

Now we consider the third claim. Suppose the algorithm has already recursed on Y. Since $C \cap c = \emptyset$, P_{11} again shows that $Y_{*,c} = 0$. As argued above, $Y'' = Y$, so $\hat{Y}_{*,c} = Y_{*,c} = 0$. Thus $Z'' = Z - \hat{Y}_{*,c}(d_{r,c})^{-1}\hat{X}_{r,*} = Z$, as required. A symmetric argument holds if the algorithm has already recursed on X. ■

Claim A.6. *P_{11} holds after the update of Eq.* (A.5).

Proof. We just consider columns; the argument for rows is identical. Consider some submatrix in $\{W, X, Y, Z\}$, let M be the value of that submatrix before recursing on W, and let M'' be the value after recursing on W. If $M = W$, then it is clear from Eq. (A.4) that either $M''_{*,j} = 0$ or $j \in c$, as required. Now suppose that $M \neq W$ and the algorithm has already recursed on M before recursing on W. Since P_{11} held before recursing on W, we have either $M_{*,j} = 0$ or $j \in C$, as required. By Claim A.5, $M'' = M$, so the claim follows. ■

Finally, we show that property P_h holds after the algorithm has recursed on every submatrix. Let $i \notin R$ be arbitrary. By P_{11}, the entries of this row are zero in

both submatrices containing this row. Thus $D_{\overline{R},*} = 0$, as required.

A.5 Inversion, etc.

Given the factorization of A satisfying $H(A, L, D, U, R, C)$, we now explain how to compute a maximum-rank submatrix, to compute the determinant, and to invert A if it is non-singular.

First, $\operatorname{rank} A$ is simply $|R|$; this holds by P_4, P_5 and P_h, and since L and U are non-singular. Furthermore, $A_{R,C}$ is a maximum-rank submatrix of A. If $|R| < n$ then $\det A = 0$. Otherwise, let π be the permutation corresponding to the generalized permutation matrix D. Then $\det A$ is the product of the non-zero entries of D, multiplied by $\operatorname{sign}(\pi)$.

Now suppose that A is non-singular, and thus D is also. We have

$$D = L \cdot A \cdot U \quad \Longrightarrow \quad I = U \cdot D^{-1} \cdot L \cdot A.$$

Since D is a generalized permutation matrix, it can be inverted in $O(n^2)$ time. Thus $A^{-1} = U \cdot D^{-1} \cdot L$ can be computed in $O(n^\omega)$ time.

Bibliography

[1] Martin Aigner and Thomas A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, July 1971.

[2] Michael Artin. *Algebra*. Prentice Hall, 1991.

[3] Hélène Barcelo and Arun Ram. Combinatorial representation theory. In Louis J. Billera, Anders Björner, Curtis Greene, Rodica Simion, and Richard P. Stanley, editors, *New perspectives in algebraic combinatorics*, volume 38 of *Mathematical Sciences Research Institute Publications*, pages 23–90. Cambridge University Press, 1999.

[4] Alexander I. Barvinok. New algorithms for linear k-matroid intersection and matroid k-parity problems. *Mathematical Programming*, 69:449–470, 1995.

[5] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[6] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, 1967.

[7] Thomas H. Brylawski. An affine representation for transversal geometries. *Studies in Applied Mathematics*, 54(2):143–160, 1975.

[8] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[9] P. M. Camerini, G. Galbiati, and F. Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

[10] Amit Chakrabarti. Lower bounds for multi-player pointer jumping. In *Proceedings of the 23rd IEEE Conference on Computational Complexity (CCC)*, pages 33–45, 2007.

[11] Prasad Chalasani and Rajeev Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 26(6):2133–2149, 1999.

[12] Joseph Cheriyan. Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, 26(6):1635–1669, 1997.

[13] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

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

[15] Don Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13(1):42–49, 1997.

[16] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

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

[18] William H. Cunningham. On submodular function minimization. *Combinatorica*, 3:185–192, 1985.

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

[20] Carsten Damm, Stasys Jukna, and Jiri Sgall. Some bounds on multiparty communication complexity of pointer jumping. *Computational Complexity*, 7(2):109–127, 1998.

[21] Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[22] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[23] Jack Edmonds. Matroid partition. In G. B. Dantzig and A. F. Veinott Jr., editors, *Mathematics of the Decision Sciences Part 1*, volume 11 of *Lectures in Applied Mathematics*, pages 335–345. American Mathematical Society, 1968.

[24] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications*, pages 69–87. Gordon and Breach, 1970.

[25] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[26] Jack Edmonds. Matroid intersection. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 39–49. North-Holland, 1979.

[27] Shimon Even and Oded Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 100–112, 1975.

[28] M. Fujii, T. Kasami, and K. Ninomiya. Optimal sequencing of two equivalent processors. *SIAM Journal of Applied Mathematics*, 17:784–789, 1969.

[29] Satoru Fujishige. Polymatroidal dependence structure of a set of random variables. *Information and Control*, 39:55–72, 1978.

[30] Satoru Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, second edition, 2005.

[31] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 112–122, 1991.

[32] Harold N. Gabow and Ying Xu. Efficient algorithms for independent assignments on graphic and linear matroids. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 106–111, 1989.

[33] 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.

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

[35] Chris D. Godsil. *Algebraic Combinatorics*. Chapman & Hall, 1993.

[36] Michel X. Goemans. Bounded degree minimum spanning trees. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[37] Andrew V. Goldberg and Alexander V. Karzanov. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, 100(3):537–568, July 2004.

[38] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[39] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[40] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, second edition, 1993.

[41] András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 186–191, 1988.

[42] 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.

[43] Refael Hassin and Asaf Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing*, 33(2):261–268, 2004.

[44] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[45] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n - o(n) for boolean circuits. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, pages 353–364, 2002.

[46] Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, March 2008.

[47] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. *Journal of the ACM*, 48:761–777, 2001.

[48] Gordon James and Adalbert Kerber. *The Representation Theory of the Symmetric Group*. Addison-Wesley, 1981.

[49] David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.

[50] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

[51] Howard J. Karloff. A Las Vegas RNC algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[52] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 759–770, 1992.

[53] Donald E. Knuth. The asymptotic number of geometries. *Journal of Combinatorial Theory, Series A*, 16:398–400, 1974.

[54] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[55] Eugene L. Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9:31–56, 1975.

[56] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, 2001.

[57] László Lovász. On two minimax theorems in graph. *Journal of Combinatorial Theory, Series B*, 21:96–103, 1976.

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

[59] László Lovász. Submodular functions and convexity. In Achim Bachem, Martin Grötschel, and Bernhard Korte, editors, *Mathematical Programming: The State of the Art*, pages 235–257. Springer, 1983.

[60] László Lovász. Communication complexity: A survey. In B. H. Korte, editor, *Paths, Flows, and VLSI Layout*, pages 235–265. Springer Verlag, 1990.

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

[62] László Lovász and Michael E. Saks. Communication complexity and combinatorial lattice theory. *Journal of Computer and System Sciences*, 47(2):322–349, 1993.

[63] S. Thomas McCormick. Submodular function minimization. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 321–391. North-Holland, 2005.

[64] N. Meggido and A. Tamir. An $O(n \log n)$ algorithm for a class of matching problems. *SIAM Journal on Computing*, 7:154–157, 1978.

[65] Kurt Mehlhorn and Erik Schmidt. Las vegas is better than determinism in vlsi and distributed computing. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–337, 1982.

[66] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

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

[68] Marcin Mucha. *Finding Maximum Matchings via Gaussian Elimination*. PhD thesis, Warsaw University, 2005.

[69] 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.

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

[71] Kazuo Murota. *Discrete Convex Analysis*. SIAM, 2003.

[72] Mark Aronovich Naĭmark. *Theory of Group Representations*. Springer-Verlag, 1982.

[73] H. Narayanan, Huzur Saran, and Vijay V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arboresences and edge-disjoint spanning trees. *SIAM Journal on Computing*, 23(2):387–397, 1994.

[74] James B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*. To appear in Mathematical Programming. DOI 10.1007/s10107-007-0189-2.

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

[76] Christos H. Papadimitriou and Michael Sipser. Computational complexity. *Journal of Computer and System Sciences*, 28(2):260–269, 1984.

[77] Stephen Ponzio, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. The communication complexity of pointer chasing. *Journal of Compter and System Sciences*, 62(2):323–355, 2001.

[78] Michael O. Rabin and Vijay V. Vazirani. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10(4):557–567, 1989.

[79] Arun Ram. Characters of Brauer's centralizer algebras. *Pacific Journal of Mathematics*, 169(1):173–200, 1995.

[80] Ran Raz and Boris Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15(4):567–588, 1995.

[81] Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

[82] Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3(3):371–384, 1976.

[83] Bruce E. Sagan. *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*. Springer, second edition, 2001.

[84] Piotr Sankowski. Processor efficient parallel matching. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 165–170, 2005.

[85] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80:346–355, 2000.

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

[87] Richard P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 1999.

[88] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[89] Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. Manuscript, 2008.

[90] Nobuaki Tomizawa and Masao Iri. An Algorithm for Determining the Rank of a Triple Matrix Product AXB with Application to the Problem of Discerning the Existence of the Unique Solution in a Network. *Electronics and Communications in Japan (Scripta Electronica Japonica II)*, 57(11):50–57, November 1974.

[91] Donald M. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.

[92] William T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947.

[93] Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph matching algorithm. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 509–530, 1990.

[94] Dominic J. A. Welsh. *Matroid Theory*, volume 8 of *London Mathematical Society Monographs*. Academic Press, 1976.

[95] Hassler Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.

[96] Marcel Wild. The asymptotic number of inequivalent binary codes and nonisomorphic binary matroids. *Finite Fields and their Applications*, 6:192–202, 2000.